NPS52-88-013

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A COMPUTER SIMULATION STUDY OF STATION KEEPING
BY AN AUTONOMOUS SUBMERSIBLE
USING BOTTOM-TRACKING SONAR

by

Chet A Hartley

June 1988

Thesis Advisor: Robert B. McGhee

Prepared for:
Naval Postgraduate School
Monterey, California 93943-5000

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. C. Austin
Superintendent

Kneale T. Marshall
Acting Provost

This thesis is prepared in conjunction with research funded by the Naval Postgraduate School under the cognizance of the Naval Surface Weapons Center, White Oak.

The issuance of this thesis as a technical report is concurred by:

ROBERT B. MCGHEE
Professor
of Computer Science

Reviewed by:

VINCENT Y. LUM
Chairman
Department of Computer Science

Released by:

JAMES M. FREMGEN
Acting Dean of Information and
Policy Science

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NPS52-88-013	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b. OFFICE SYMBOL (If applicable) Code 52
7a. NAME OF MONITORING ORGANIZATION Naval Surface Weapons Center, White Oak	
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Silver Spring, MD 20903
8a. NAME OF FUNDING SPONSORING ORGANIZATION Naval Postgraduate School	8b. OFFICE SYMBOL (If applicable)
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER O & MN, Direct Funding	
8c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)
A Computer Simulation Study of Station Keeping by an Autonomous Submersible Using Bottom-Tracking Sonar

12. PERSONAL AUTHOR(S)
Hartley, Chet A

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM _____ TO _____	1988 June	109

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Artificial Intelligence; Robotics; Graphics; Autonomous Underwater Vehicle; Inertial Navigation; Correlation Velocity Log Sonar; Bottom-Tracking Sonar

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

For an Autonomous Underwater Vehicle to complete many operational missions, it must have the ability to maintain its position relative to the ocean floor. Maintaining station requires that the AUV be able to determine the direction and the distance displaced during a small time interval. Knowing the direction and distance traveled in a measured amount of time, the magnitude and direction of the ocean current can be calculated. Once this ocean current information is known, the AUV speed and direction can be properly adjusted to directly offset the ocean current forces.

This thesis will attempt to determine, by computer simulation, if the first problem of AUV station keeping, vehicle movement direction and distance detection can be performed using bottom-tracking sonar as the AUV's only sensor. Both the problems of performing and storing successive synthetic sonar images and of determining AUV motion using frame to frame correlation of these images are investigated.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Robert B. McGhee	22b. TELEPHONE (Include Area Code) (408) 646-2095 / 22c. OFFICE SYMBOL Code 52Mz

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

U.S. Government Printing Office: 1986—606-243

Approved for public release; distribution is unlimited.

A Computer Simulation Study of Station Keeping by an Autonomous Submersible Using Bottom-Tracking Sonar

by

Chet A Hartley
Lieutenant, United States Coast Guard
B.S., United States Coast Guard Academy, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

June 1988

Author: _____
Chet A Hartley

Approved by: _____
Robert B. McGhee, Thesis Advisor

_____
Roberto Christi, Second Reader

_____
Vincent Y. Lum, Chairman
Department of Computer Science

_____
James M. Fremgen,
Acting Dean of Information and Policy Sciences

ABSTRACT

For an Autonomous Underwater Vehicle to complete many operational missions, it must have the ability to maintain its position relative to the ocean floor. Maintaining station requires that the AUV be able to determine the direction and the distance displaced during a small time interval. Knowing the direction and distance traveled in a measured amount of time, the magnitude and direction of the ocean current can be calculated. Once this ocean current information is known, the AUV speed and direction can be properly adjusted to directly offset the ocean current forces.

This thesis will attempt to determine, by computer simulation, if the first problem of AUV station keeping, vehicle movement direction and distance detection can be performed using bottom-tracking sonar as the AUV's only sensor. Both the problems of performing and storing successive synthetic sonar images and of determining AUV motion using frame to frame correlation of these images are investigated.

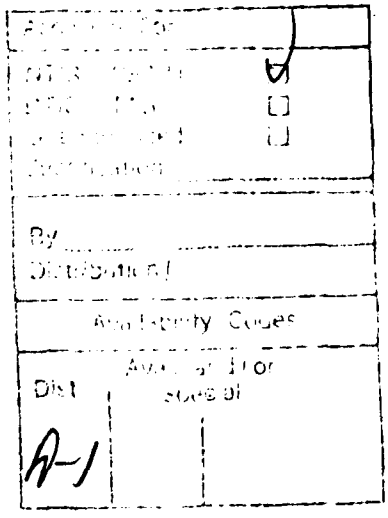


TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

This research could not have been completed without the guidance and support of my advisor, Professor Robert B. McGhee. Professor McGhee would willingly take time from his very busy schedule to discuss problems I had, always offering pertinent advice that would shed just enough light on the problem to relieve my frustration, but not give away the answer. He made learning interesting and fun. My appreciation also goes to Professor Roberto Christi for agreeing "sight unseen" to take the time and effort necessary for being my second reader.

Finally, I am most grateful to my wife, Dana, for her understanding, patience and support during my studies and throughout our entire marriage. For my son, Christopher, whenever I came home from a tough day at school, your happy face and sense of humor gave me a much needed chance to smile.

I. INTRODUCTION

A. GOALS

The goal of this thesis is to show that an Autonomous Underwater Vehicle (AUV), can maintain a constant position over an arbitrary location on the ocean floor. To perform this station keeping operation, the AUV must first be able to detect its own motion. Since current motion detection devices are either accurate and large or inaccurate and small, this thesis aims to find a small and accurate motion sensor. This research work explores the idea of using bottom-tracking sonar as the small and accurate motion detector.

In order to conduct an initial feasibility study of this idea, it is desireable to construct a computer simulation of an actual bottom-tracking sonar. Once the sonar simulator is built, AUV motion control can be incorporated. During this portion of the project, the *environmental* machinery should be developed in order to allow AUV course, speed and dive angle control along with ocean current magnitude and direction manipulation. If these two portions can be completed, the sonar model will be operating within a simulated, alterable environment. These two steps are accomplished in this thesis.

The final portion of the present research involves construction of the model necessary to conduct successive scans of the ocean floor, compare them, and determine from the comparison, the direction and distance traveled between scans. Successful realization of this goal provides the basis for design of a station keeping autopilot in subsequent research. Only if all three of these research portions are completed, can the goal of this thesis be tested and met.

B. ORGANIZATION

Chapter II presents a discussion of work dealing with early AUV systems as well as projects currently under development. Also work on sonar systems as well as inertial reference systems is discussed.

Chapter III contains a more detailed problem definition along with a discussion on how the sonar data is mathematically manipulated to obtain the x, y and depth coordinates of the ocean floor terrain being scanned. A discussion is presented on how two successive bottom scans are compared, in order to detect AUV motion, and finally, a description of the simulation suite is presented.

The entire system simulation model is broken down by module and a description of each is presented in Chapter IV. The model used for the ocean floor terrain is described in more detail and the simulation program User's manual is also included.

Chapter V focuses on two simulation experiments. A discussion on how selective changes in the simulated environment effect simulation results is presented. The last chapter, Chapter VI, presents some conclusions about the work described in and research contributions of this thesis, followed by recommendations for possible future work.

II. REVIEW OF PREVIOUS WORK

A. INTRODUCTION

Since this thesis models an Autonomous Underwater Vehicle (AUV) that uses a bottom-scanning sonar, brief histories of the development of both Unmanned Underwater Vehicles and sonar systems are presented in this chapter. Descriptions of five Remotely Operated Vehicles (ROVs) and two AUVs are included. A description and a brief history of Inertial Navigation Systems (INS) is also presented.

B. UNDERWATER VEHICLES

1. Brief History

Prior to the mid 1960's, almost all subsea tasks could be completed by manned submarines or ROVs, but manned submarines were limited in endurance and ROVs were limited in range. As new missions or specific tasks evolved demanding more endurance and range, AUVs started to attract attention. Actually, interest in AUVs for military and industrial applications began in the early 1960's, but manned submersibles and ROVs captured most of the attention from 1965 to the recent past. As stated above, manned submersibles were limited in endurance and could not operate in conditions hazardous to humans. ROVs were developed to increase this endurance and to have the capability to operate in high sea states or environmentally hazardous surroundings. [Ref. 1]

ROVs were and still are well suited for many different situations:

- Diver replacement for inspection at depths too deep for human divers
- Carrier for test of underwater equipment
- Carrier of training equipment for Navy use
- Sensor vehicle used either singularly or in conjunction with ships or subs

- Independently patrolling vehicle with information processing capabilities
- Unmanned sensor and weapon platform for short-range operations
- Coast guarding operations
- Rescue operations
- Monitoring of water quality
- Mining operations on the sea floor.

However, there are drawbacks to ROVs. Some drawbacks to tethered ROVs are listed below:

- The tether is a constraint in operations
- The surface support ship requires accurate station keeping
- ROVs are sensitive to the ocean current
- There is a necessity for a tether management system.

Although untethered ROVs eliminate the drawbacks caused by the tether, they, along with tethered ROVs, require a human operator. Since the early ROVs were controlled by a master-slave configuration, none of the vehicle tasks were automated. In order to reduce costs and personnel risks, and increase overall system performance, these undersea vehicle tasks began to be automated. System architectures and/or supervisory controls were used to automate previously manual tasks and this means of control was adopted for developing advanced untethered submersibles. With the advent of microprocessors, vehicles, with embedded microprocessors, could be configured to perform a defined task on command. High level commands could be issued by the operator to complete a task and thereafter the operator would only be required to supervise task execution. [Ref. 2]

As ROV technology matured into very reliable systems and supervisory control became more high level, the concept of AUVs became more appealing. An autonomous vehicle is considered to be an advanced supervisory controlled vehicle which has been extended with mission level command capability, meaning that the vehicle can be

commanded to automatically perform complete mission tasks without any human operator intervention. Some of the initial AUV applications are listed below:

- Extended distance reconnaissance and search operations
- Under-ice operations
- Inspection and supervision of poison disposals or explosive devices dumped at sea
- Underwater structure operations where tethered vehicles can not operate.

Interest in AUVs was enhanced in the mid 1970's by technological advances in micro-electronics, high-speed digital computers, artificial intelligence, navigation, guidance and control hardware, logic and software. The AUV advances made during this period, generated mainly by the offshore oil industry and the military, emerged in the early 1980's in a new breed of sophisticated torpedo targets, decoys and countermeasures, and special purpose AUVs. The list of organizations that had built AUVs had grown substantially and expanded internationally by the mid 1980's. [Ref. 1]

The seemingly simple move from ROVs to AUVs created new functional problems. *Everything* must be taken care of by the vehicle itself, the most important of which are listed below:

- Intelligence
- Communications
- Navigation
- Energy
- Handling and support systems.

Some of these areas are well developed and the technology to perform them is immediately available. Other areas such as intelligence and communications are still evolving. [Ref. 2]

2. Remotely Operated Vehicles

a. The SUTEC SeaOwl

The **SUTEC SeaOwl** is owned and operated by the Swedish Navy. It is used as a general purpose ROV, most often as a support to divers and sometimes a replacement for using divers. It has also been used for underwater inspections, torpedo recovery (up to depths of 150 meters) and search operations. The Swedish Navy also operates the **SUTEC UVEN** which is used for mine destruction while operated from a minehunting vessel. [Ref. 2]

b. The DOLPHIN

The **DOLPHIN** (Deep Ocean Logging Platform with Hydrographic Instrumentation for Navigation) multi-vehicle computer controlled system is an unmanned, untethered semi-submersible designed as a stable platform for hydrographic research. When it was developed in 1983 for the Canadian Hydrographic Service, it consisted of one vehicle controlled by a single operator console over a real time radio link. Since then, the **DOLPHIN** system has expanded to eight vehicles controlled by three consoles. The **DOLPHIN** vehicles have automated launch/recovery, positioning and course following, and fault tolerant software. [Ref. 3]

The **DOLPHIN** vehicle itself, created to replace a three man launch which was limited to daylight hour and low sea state operations, is a radio-controlled, snorkelling, diesel powered semi-submersible. It is capable of speeds up to 16 knots at depths of 10 to 15 feet in seas up to sea state 6. The U.S. Navy purchased two **DOLPHIN**-type vehicles in 1986 for evaluation as possible countermeasure platforms [Ref. 4].

c. The ARCS

The Canadian Hydrographic Service also sponsored the development of **ARCS** (Autonomous Remote Controlled Submersible). Like the **DOLPHIN**, International Submarine Engineering of Canada, built the **ARCS** to cost-effectively chart waters normally covered by ice. **ARCS** is an untethered autonomous remotely operated vehicle that operates through a hole in the ice and has a high level of reliability. The **ARCS** is lowered through a hole in the ice, travels along a preprogrammed grid while storing depth, altitude and position data on magnetic tape, and returns to the drop hole for recovery at grid completion. The **ARCS** has novel operator interfaces and a high degree of onboard intelligence. It is torpedo shaped and weighs approximately 1820 kg. It carries an obstacle avoidance sonar, an acoustic altimeter, a depth meter, a two axis doppler sonar, an acoustic long baseline positioning system and is driven by a single propeller at the stern. [Refs. 1, 3]

d. The AUSS

The U.S. Navy has developed an untethered sensor search vehicle called the **AUSS** (Advanced Unmanned Search System). The **AUSS** is battery-powered vehicle that carries a search sensor suite consisting of a forward-looking and a side-scan sonar as well as a video camera and a 35mm still camera. It also carries a transponder for use in a long-baseline acoustic navigation field and a receiving/transmitting transducer. When deployed, the **AUSS** follows a pre-programmed search pattern contained in its on-board computer which can be adjusted by commands sent by the operator via the acoustic telemetry system. The on-board doppler speed sensor and gyrocompass generate information used for navigation. The surface operator can view the video images generated by the sonars/TV camera during the search and can interrupt the search pattern to investigate interesting objects. After photographing and inspecting these objects, the

AUSS will return to its automatic search mode. This process continues until the vehicle needs a battery charge or until the search is completed. [Ref. 5:pp. 1-2]

e. The ATV

The U.S. Navy has also developed a new work system called the ATV (Advanced Tethered Vehicle) which uses new advances in fiber optic technology to reduce the tether size and weight. The ATV is a tethered vehicle that uses thrusters for propulsion, has control devices, manipulators, tools and a TV viewing system. The tether transmits electrical power as well as video information. The first of three manipulators on the ATV is a simple unit used for gripping. The other two are seven-degree-of-freedom units intended to do complex work tasks, both directly and with tools. The TV system transmits close-up pictures of the work to the operators on the surface and includes a stereo pair of cameras on a pan-and-tilt platform, a single camera with a zoom lens, and a fixed camera for position reference. When deployed, the vehicle is driven to the bottom and navigated to the work site. The preliminary approach to the work site uses the forward-looking sonar. The TV cameras are then used to more accurately position the ATV and to monitor the work itself. When all work is completed, the system is driven clear of the work site and to the surface and then retrieved. [Ref. 5:pp. 3-4]

3. Autonomous Underwater Vehicles

a. The EPAULARD

The **EPAULARD**, operational since 1981, was developed by IFREMER of France for systematic undersea exploration missions, mainly deep sea nodule exploration. It is capable of performing high quality deep sea systematic photographic imaging and micro-bathymetry and has been used for underwater canyon surveys and

photographic surveys of radioactive waste dumping areas. It has been pioneering AUV application research in geology, marine biology, and searching for wrecks. [Ref. 6]

b. The KB/EAVE

The **KB/EAVE** (Knowledge-Based/Experimental Auto Vehicle) is a third generation autonomous untethered submersible utilizing state-of-the-art processors, operating systems, sensors and artificial intelligence concepts. It was built by the Marine Systems Engineering Laboratory, University of New Hampshire. It is an open framed vehicle, providing great flexibility for adding and controlling sensors and effectors, that has a primary mission as a development testbed for new technology. It is highly maneuverable and can control on altitude (terrain following) as well as on depth. It contains vertical, slide, and forward thrusters, navigation, depth and altitude sonars, navigation guidance control, and an obstacle avoidance radar. It is 50 x 41 x 45 inches and weighs 1000 pounds. Its maximum speed is 1.5 knots and can reach depths up to 500 feet. [Ref. 7]

C. SONAR SYSTEMS

1. Brief History

At the start of World War I the Royal Navy began to develop a weapon to combat the devastating new German weapon: the U-boat. Traditional naval methods were of little use against this menace, so existing technology was used to produce hydrophones (underwater microphones). These hydrophones were used extensively during the war but were not very effective. Soon after the first World War, research efforts switched to developing an active echo-ranging device, which the British called **ASDICS** (Allied Submarine Detection Investigation Committee), rather than the passive hydrophones. It was determined that an active echo-ranging system would allow surface ships to detect, attack and destroy enemy submarines much more effectively. [Ref. 8]

By the start of the second World War, the Royal Navy had an operational anti-submarine **ASDICS** system available, based on the original World War I quartz transducer, housed in a streamlined dome and connected to a range recorder. The dome and the range recorder were important innovations because the dome allowed the **ASDIC** set to be used up to ship speeds of twenty knots, and the range recorder plotted the changing ranges of the target during the attack. [Ref. 8]

Virtually nothing was known about the behavior of acoustic waves in water when research into the military use of underwater sound was started. Initially, optical theory and the behavior of sound waves in air were used as models. After research, it was discovered that sound moves four times faster through water than through air, but its exact speed varies slightly with temperature, pressure and salinity. It was also quickly and painfully discovered that the ocean was, by no stretch of the imagination, a perfect acoustic medium. It was found that as a sound pulse travels away from the transducer, some of the energy is lost through spreading, some through scattering, and a small amount is converted into head because of the viscosity of the water. It was also found that water conditions distorted the sound beam, mainly because of refraction, but also from reflection. Reflection from the surface was found to be almost total, but from the sea bed was much more complex. Early researchers had to contend with these and many more problems. [Ref. 8]

The backbone of the first echo-ranging devices was the electro-acoustic transducer, which converted an electrical signal into an acoustic signal and vice-versa. Thus the same device was used for sound transmission and reception. R. A. Fessenden developed the first successful transducer using an electrodynamic oscillator (patented in 1913), but this transducer operated at a low sonic frequency. The first ultrasonic transducer was the quartz-steel sandwich of Langevin and Chilowski, which formed the

basis of the Royal Navy's ASDIC transducer used in the second World War. Electrostrictive ferroelectric transducers such as those of barium titanate and lead zirconate began to be developed in the late 1940's and early 1950's and are used extensively today. The main advantage of these new materials is that they can be manufactured to specific requirements and are more efficient, cost effective and more durable than the older transducers. [Ref. 8]

During and after the first World War, underwater whistles, jets, and water-hammers were investigated as underwater sound sources for signaling and echo-ranging. More effective hydroacoustic sources began to be developed in the late 1960's and were far more successful than their primative predecessors. Transducers behave in water like rapid moving pistons. Their design is highly complex and in order to locate a target accurately, a narrow sound beam is required. A narrow sound beam can only be constructed by using a transducer whose vibrating surface is many times the wavelength of the sound beam. During the second World War, even higher resolution was achieved by shortening the transmitted sound pulse. [Ref. 8]

A new name for these echo-ranging systems was coined in 1942 by the American underwater acoustics specialist and Director of the Wartime Harvard Underwater Sound Lab, F. V. Hunt. This new name was **SONAR**, for Sound Navigation and Ranging. The term **SONAR** had originally only related to echo ranging equipment, but in 1943 it was defined as *the science and the art of transmission and reception of underwater sound.* [Ref. 8]

The second World War was fought with mechanically rotated *searchlight* sonars. By the end of the war, systems that could detect in all directions simultaneously, known as *scanning sonars,* began to be developed and became operational in the late 1940's. These systems used much more sophisticated transducers consisting of

assemblies, or arrays, of transducer elements. Other innovations were split-beam and frequency-modulated sonars. In the postwar period large and complex transducer arrays of different configurations operating over a wide frequency band were developed. Also, transducers with electronically steered acoustic beams of different shapes for particular directional effects, and of great acoustic power were developed. Technological advances have led to a lowering of the operating frequencies and an increase in range. In 1919 the average echo range was about 500 yards. During the second World War, it increased to 1300 yards and at present is several miles. [Ref. 8]

After World War I and before World War II, the German Navy was the only significant navy in the world developing the hydrophone arrays or *passive* (listening) sonar. All other major navies discarded the hydrophone in favor of the *active* (sound producing) sonars. After the war, the Allies found the passive systems to be very effective and this passive sonar technology was taken up by the West and by the Soviet Union. This new sonar technology began to reach the ships in the early 1950's. [Ref. 8]

The birth of the nuclear submarine, with its almost unlimited endurance and high underwater speed has resulted in a reappraisal of existing sonar systems and has stimulated the development of the next generation of much more sophisticated sonar systems. [Ref. 8]

2. Doppler Sonar

In the late 1960's and early 1970's a new sonar system was being developed to aid in vessel navigation, other than the common active *echosounder*. During this period, it was discovered that if a sonar beam was transmitted horizontally forward from the bow of a vessel, the received echo (reflected off of a fixed target ahead) displayed an apparent frequency shift from the originally transmitted one. This frequency shift was proportional to the vehicle's velocity. This frequency shift, called the *Doppler effect*, is

common to all forms of radiation and was the basis for the development of the **DOPPLER** sonar. The relationship between velocity and frequency shift is given by

$$\Delta f = \frac{2Vf_o}{S} \tag{2.1}$$

where Δf is the transmitted and received frequency difference *(Doppler frequency)* , V is the velocity of the vessel, f_o is the originally transmitted frequency, and S is the sound propagation velocity. [Ref. 9:p. 298]

The Doppler sonar was very effective if there were stationary reflecting surfaces directly ahead of the vessel at approximately the same depth as the transducer. Since no shallow reflecting surfaces usually exist ahead of a vessel, the sonar beam was directed downwards to obtain a fixed reference point. When the beam is deflected downwards at Θ^o to the vertical, the relationship between horizontal velocity and Doppler frequency becomes [Ref. 9:p. 299]:

$$\Delta f = \frac{2Vf_o \cos\Theta}{S} \ . \tag{2.2}$$

When the beam is deflected downwards, the system becomes sensitive to vertical motion. To eliminate this vertical component (usually only horizontal motion is desired), a second beam pointing aft, is deflected at the same downwards angle as the forward beam. The Doppler shift is now measured between the two beams, rather than between the transmitted and received signals. The vertical component which is common to both beams can be eliminated and a frequency shift proportional only to horizontal motion is obtained. [Ref. 9:p. 299]

The Doppler sonar system of velocity measurement, when first developed, was shown to have distinct advantages over other conventional types of vessel navigation systems. Other navigation systems at that time measured the ship's velocity relative to

the main water mass, while the Doppler sonar measured the ship's velocity over the bottom. The early Doppler systems were very accurate and were capable of measuring velocities down to 1/10 of a knot. Obviously the Doppler sonar is very effective when the water is shallow enough for a bottom echo to be received. Surprisingly though, the Doppler system has also shown adequate performance in making use of the mid-water reverberation (sound reflecting "water layer") return. Even though using these reverberation layers does not eliminate the effect of underwater currents on vessel velocity determination, substantial accuracy is still maintained because shallow ship disturbances and surface current errors are eliminated. [Ref. 9:p. 299]

Four Doppler sonar beams, pointing fore, aft and out from each beam can produce a more sophisticated navigation aid that will measure vessel velocity in those four directions. A very accurate measure of the ship's velocity over the ground can then be obtained independently of heading by adding the for/aft and athwartships velocities vectorially. [Ref. 9:p. 301]

3. Correlation Velocity Log Sonar

In the late 1970's, a new process to measure a ship's velocity was developed. The *Correlation Velocity Log* sonar system transmits sound into the water in the same manner as a conventional depth sounder, but uses an array of hydrophones to receive return signals. With each transmission, each of the hydrophones contained in the *CVL* array receives a unique time waveform that has been randomly dispersed by the ocean bottom. Cross-correlation measurements among the various hydrophones for the return from pairs of transmitted pulses are analyzed to locate the peaks of the correlation functions. The peaks of these response functions relate directly to the distance traveled by the vessel during the interpulse interval. The vessel's velocity and distance traveled can then be calculated using this and time lapse information. This new CVL sonar

system, currently being produced by the Undersea Systems Department of the General Electric Company, is simpler than the Doppler sonar system. The CVL requires significantly less total acoustic energy dispersion, is optimally matched to the ocean environment and sound-velocity variation calibrations are not required. [Ref. 10]

The General Electric CVL model **QV-12** Sonar System measures a vessel's speed relative to the ocean bottom under a wide range of operating environments. It was designed to provide bottom referenced velocities in water depths up to 5,000 meters. The transducer is only 10.7 inches in diameter and 8 inches high, while the electronics package is 14 x 19 x 20 inches. The hydrophones in the CVL array are arranged in a *U* shape in order to detect both fore/aft and athwartship vessel velocity. The **QV-12** was also designed to achieve total autonomy; a stand-alone navigation device with no external control required while operating over a wide variety of depths, speeds, and sea beds. One very interesting feature of the **QV-12** is that it will automatically switch to a *water tracking* mode whenever the bottom cannot be tracked. While in the *water tracking* mode, the **QV-12** will periodically test if the bottom can be tracked, and will reacquire bottom tracking as soon as it is possible to do so. [Ref. 11]

The CVL system can provide a true earth referenced velocity input to an integrated navigation or positioning system in water depths that far exceed the maximum bottom-tracking range of conventional Doppler sonar. Reported performance of this instrument is better than 0.4 percent of vehicle velocity and can operate in depths below the keel of 20,000 feet. The CVL uses a single vertically downward looking broad beam, 40° beamwidth, which permits the use of a low operating frequency, 18kHz. This broad beam provides good coverage of the bottom even with large changes in vessel pitch and roll during beam propagation. [Ref. 12]

With the present CVL technology, a vessel's bottom referenced velocity can be measured in virtually any water depth. The CVLs have greater accuracy and dependability than Doppler sonars because of the elimination of bias errors attributable to changes in bottom characteristics and to the use of inaccurate values of local sound speed. [Ref. 13]

D. INERTIAL NAVIGATION SYSTEMS

1. Fundamentals of Inertial Navigation

Navigation is the process of directing the movement of a vehicle form one point to another. Navigation involves the determination and indication of the position, attitude, and velocity of the vehicle relative to reference coordinates at a given moment [Ref. 14]. During the early 1940's a new automatic navigation technique, inertial guidance, was developed to guide missiles to their targets. This new navigation technique offered several significant advantages over previous techniques:

- Invulnerability to enemy jamming
- No ground facilities required
- No radiation emitted, preventing enemy detection.

These first inertial navigation systems weighed anywhere from 75 to 2,000 pounds, depending upon intended use, required accuracy, and duration of the mission. Many of the basic inertial system components, such as gyros, accelerometers, computers and servo systems, had been used aboard aircraft, but were a far cry from those required for inertial navigation systems. The instrumentation accuracy, needed to give inertial guidance systems even moderately acceptable performance was equivalent to the extreme precision normally found only in research laboratory standards. An vehicle mounted inertial guidance system, however, must operate in the extremes of temperature, vibration and shock from the vehicle. [Ref. 15:p. 4]

Any inertial navigation system is essentially a form of dead-reckoning device. This means that the geographic position (latitude and longitude or equivalent) of both the starting point and destination must be known and set into the equipment. Once in operation, the INS is capable of determining the following information:

- Geographic position of the vehicle
- Ground velocity
- Distance traveled
- Direction to destination
- Attitude of the vehicle.

A pure inertial navigation system relies on Newton's laws of motion, primarily that a body at rest will remain at rest unless some external force is applied to it, and that a body in motion will continue to move in a straight line unless some force acts on it. The inertial system is able to determine the displacement of the carrying vehicle from its starting point by measuring the accelerations of the vehicle relative to the earth. The forces acting on the vehicle and its inertial instrumentation make it possible to measure vehicle accelerations. [Ref. 16:p. 26]

Velocity is the rate of change of distance with time, and acceleration is the rate of change of velocity with time. If a vehicle's acceleration relative to the earth can be measured, a double integration (Equation 2.1) of this acceleration over a time interval will give the total distance which the vehicle has traveled during that interval.

$$Distance\text{-}traveled = \iint Acceleration(t)\, dt^2 \tag{2.3}$$

With this distance traveled information, a continuous navigational track independent of any outside information can be maintained. Inertial Navigation Systems are not perfect, and errors accumulate with time, consequently, external navigation fixes must be used to *correct* or *reset* the INS from time to time depending upon the accuracy of the INS. [Ref. 15:p. 5]

Regardless of the type of engineering implementation used, all inertial navigation systems must perform the following functions:

- Establish and maintain a reference frame of coordinates
- Measure specific forces applied
- Have knowledge of the gravitational field
- Time integrate the specific force data to obtain velocity and position information

Three appropriately located gyroscopes can form the basis for establishing and maintaining a reference frame of coordinates for the INS. The forces applied to the vehicle are measured using accelerometers. Most accelerometers use a pendulum system where the motion of the pendulum can be directly related to the vehicle's acceleration. A detailed knowledge of the local gravitational field is necessary for the accelerometer to distinguish between inertial and gravitation accelerations. This information is a must if an inertial position is to be found with sufficient accuracy. [Ref. 16:pp. 26-27]

2. Brief History

Inertial guidance as a practical art for controlling vehicle motion between points on the earth did not exist before 1940. All the impetus to start development in this area stemmed directly from military requirements generated by World War II. The Peenemünde group of German Scientists who developed the V-2 ballistic rocket missile (1942) receive the credit for the realization of inertial guidance. The guidance and control system for the V-2 rocket employed two two-degree-of-freedom gyros, one to control rolling and yawing motion and the other to control pitch. The accelerometer in the system consisted of a gyroscopic pendulum mounted in a gimbal. As the pendulum deviated from its neutral position, it switched on an induction motor which rotated the gimbal so that the pendulum was caused to precess to its neutral position. The speed of the missile was then obtained by an appropriate recording technique. At the end of

World War II, the German group had a stable platform using three single-degree-of-freedom gyros, employed in conjunction with an integrating accelerometer to take into account the inclination of the flight path to the vertical. [Ref. 17:pp. 18-19]

A number of German Scientists and engineers led by Dr. Wernher von Braun came to the United States after the end of World War II and brought with them the V-2 inertial guidance techniques and equipment. Their work continued, under sponsorship from the United States, and produced successful guidance systems for the Army **REDSTONE, JUPITER** and **PERSHING** rockets. [Ref. 18:p. 9]

American efforts in the field of inertial guidance were all started by the Air Force in the 1945-1946 period. The three problems being focused on were:

- Bombing from manned aircraft
- Delivery of warheads by subsonic winged missiles
- Delivery of warheads by supersonic winged missiles (Ballistic missiles were not yet under development).

In the beginning, American guidance developments were not directed toward completely inertial systems because of doubts that such systems could overcome the difficulties of realizing components of sufficiently high performance. These first systems used celestial body tracking to enhance guidance performance. [Ref. 17:pp. 19-20]

As experience with design, construction and tests of guidance systems accumulated, the complications of celestial body tracking were eliminated and the systems became completely inertial. Northrop Aircraft Inc., under Air Force sponsorship starting in June of 1946, successfully flight tested an INS for the **SNARK** high-subsonic-speed missile in 1954. Also in 1954, the North American Aviation Corporation, with Air Force support starting in April of 1946, successfully flight tested an INS for supersonic bombing missiles. The Instrumentation Laboratory of MIT, with

support from both the Air Force and the Navy since 1945, advanced the state of the inertial guidance art by building and testing systems of various types. [Ref. 17:p. 20]

The **FEBE** system was designed for the purpose of experimentally checking the basic principles of geometrical stabilization, floating integrating gyro units, high-performance specific force receivers, servodriven gimbals, electronic accessories, time drives, computers and may other guidance system items. The first **FEBE** system flight tests were carried out in the spring of 1949. Test runs were made between Massachusetts and New Mexico in a B-29 aircraft. Results did not indicated that **FEBE** could be considered as a production prototype, but they did show that the chances of achieving satisfactory solutions for all the problems of inertial guidance were good enough to justify further developments. [Ref. 18:p. 11]

In 1948, the U.S. Navy supported a project to design, construct and test a combination gyrocompass and stable vertical based on the principles demonstrated by the **FEBE** system. Laboratory tests of the shipboard system, called **MAST** (Marine Stable Element) were started in 1952 and were followed by shipboard tests in 1953. The preliminary tests of the system suggested that a complete inertial navigation system for naval vessels should be possible. [Ref. 18:p. 12]

On the basis of these results, developments of the Submarine Inertial Navigation System, called **SINS**, was started by the Instrumentation Laboratory in March of 1951. Results of shipboard tests completed in June of 1955 were of such quality that inertial guidance was incorporated in submarines designed to fire **POLARIS** missiles. [Ref. 18:p. 12]

Design of the **SPIRE** system (Space Inertial Reference Equipment) was started in October of 1949 under Air Force sponsorship. The first transcontinental test flight of this fully inertial system in February of 1953 was so successful that work on a system of

reduced weight and improved performance, called **SPIRE JR.**, was begun in July of 1953. This system was used on an inertially guided transcontinental flight in March of 1958 and flight testing was completed at the end of July 1958. [Ref. 17:p. 22]

In early 1954, the Instrumentation Laboratory started work on inertial guidance for ICBMs (InterContinental Ballistic Missiles). The resulting laboratory developments supplied the basis for the inertial guidance equipment manufactured for the **THOR** and the **TITAN** ICBM missiles. The background of inertial guidance work available in the Instrumentation Laboratory led the Navy to assign development and engineering test responsibility for inertial guidance systems to be used in the **POLARIS** ICBM. This work started in the fall of 1956 and was completed in 1960. [Ref. 17:p. 22]

Inertial guidance systems involve many ramifications, partly theoretical and partly practical. Work in these areas is proceeding at many places under various sponsoring agencies. There is much room for improvement and refinement of the actual equipment. It is to be expected that better performance from smaller, lighter and less-expensive systems will result from the effort that is being expended. [Ref. 17:p. 23]

E. SUMMARY

AUVs will be able to utilize the technological advances in both sonar and inertial navigation systems. INS are well suited to AUV missions involving relatively long transits, unanticipated route changes, or requiring stealth [Ref. 19]. The 1990's will undoubtedly be the decade of the AUV. This will be the period for exploitation and realization of the AUV potential in areas such as offshore, deepsea industrial and commercial applications, along with national security and scientific research.

The next chapter of this thesis will discuss the plausibility of accurate AUV station keeping using the bottom tracking sonar technology discussed earlier. Since INS will be

too large and expensive for AUV applications for some time to come, this thesis explores using a small inexpensive sonar to detect vehicle motion instead of an INS.

The Doppler and CVL sonar systems discussed earlier measure a vessel's fore/aft and athwartships velocities. When this information is used as input to an INS, an integrated navigation system is created. The information generated by the process described in this thesis, vessel direction and distance, could be an additional INS input. The INS would then have vessel velocity, direction, and distance information to maintain a high degree of accuracy over an increased length of time. This integrated system would offer the optimum combination of slow error drift, small absolute error and long intervals between position fixes.

III. PROBLEM STATEMENT AND PROPOSED SOLUTION METHOD

A. INTRODUCTION

The problem is to detect AUV motion with respect to the ocean floor. Since accurate inertial navigation systems are too large for AUV use, another method for detecting motion must be found. This research work will attempt to show, via computer simulation, that a simple and relatively small bottom-tracking sonar can detect AUV motion.

This thesis proposes to solve this motion detection problem by scanning the ocean floor with the bottom-tracking sonar, constructing a contour map of the bottom with the sonar return data, moving the AUV, scanning and creating another contour map at the new location and then comparing the two contour maps. The best match of the two contour maps can then be found, and depending on how the maps are moved relative to each other, the direction and distance between the centers of the two maps can be determined.

A simplified system explanation with illustrations is the first section in this chapter. This section is written for readers who are unfamiliar with how sonar works or who desire a non-technical description of the problem and proposed solution method presented in this thesis. The sonar being modeled, the WESMAR SS265, and the mathematical models of terrain sensing and regression analysis are described in more detail later in the chapter. A detailed description of simulation facilities is also included.

B. SIMPLIFIED SYSTEM EXPLANATION

1. How a Sonar Works

Webster defines sonar as:

> [SOund NAvigation Ranging]: an apparatus that detects the presence and location of a submerged object (as a submarine) by means of sonic and supersonic waves reflected back to it from the object.

The sonar works by transmitting and receiving sound waves. The sonar that will be simulated, the WESMAR SS265, allows the transducer, the sound transmitter and receiver, to be aimed fore, aft, port or starboard -- 360° around the vessel. The transducer can then be tilted at any angle from +4° (slightly above horizontal to the water's surface) to -90° (straight down under the vessel). The soundbeam transmits from the transducer to any section of water the user selects, much like a flashlight beam. Different sonars have different beam widths. The general rule is the narrower the beam, the better the target resolution. The wider the beam, the larger the search. Once an object is struck by the beam, it returns the resulting sound echo back to the transducer and is then converted into an electronic signal. This signal is projected onto the video monitor in a manner that can be easily interpreted. [Ref. 20]

2. Problem/Solution Overview

As stated in the introduction of this chapter, this thesis proposes to detect AUV motion by using bottom-tracking sonar to *create* and compare two consecutive contour maps of the ocean floor. If the AUV has not moved, both contour maps will be identical, and conversely, if the AUV has moved, the contour maps will be different. If portions of both maps are of the same section of the ocean floor (these map portions should be identical), one map can be laid and shifted over the other until the identical sections match. Since the AUV created these maps relative to itself, the distance and direction

that the two map centers are separated, will be the direction and distance the AUV moved between the two scans.

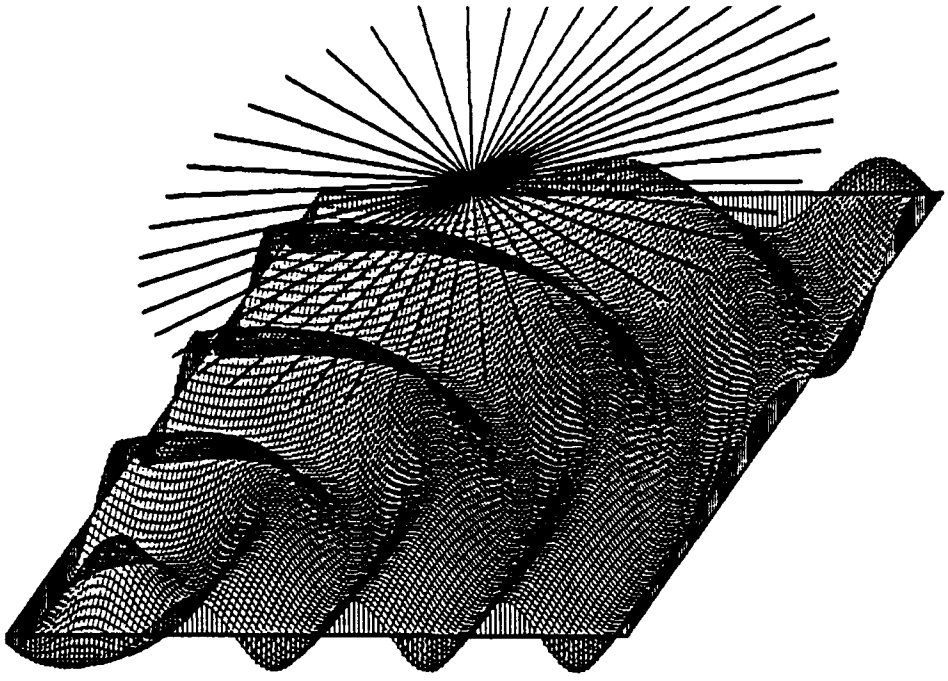
a. Contour Map Creation

To *create* these contour maps, a sonar as was described earlier, will be used. As shown in Figure 3.1*, the sonar's tilt angle, also referred to as *elevation,* will first be set to zero degrees, parallel to the ocean surface. The sweep location, also referred to as *azimuth,* will start at zero degrees and then change in eight degree increments until a 360° scan has been performed. If at any sweep location the sonar beam contacts the ocean floor, the depth below the AUV of the bottom contact will be stored in the Scan Depth array. In Figure 3.1, the sonar beam never contacts the bottom during its 360° sweep, so no depth values are stored in the Scan Depth array.

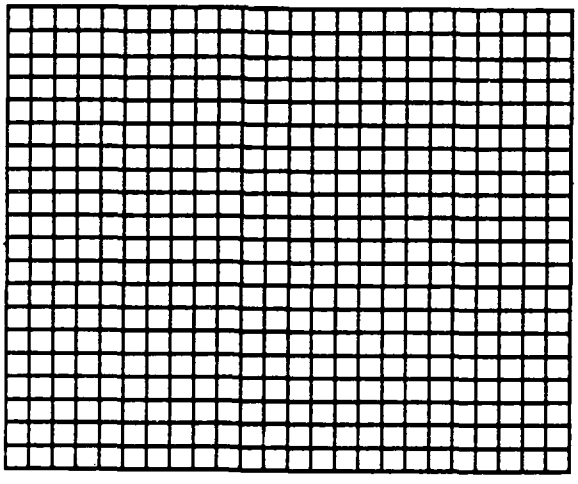
The sonar tilt angle is then decreased by an arbitrary amount. The smaller the amount, the more closely the scan depth array will resemble the actual ocean floor. For this example, the tilt angle is decreased by 30° after each 360° sweep. Figure 3.2 shows a 360° sweep with the tilt angle set at -30°. When the sonar beam contacts the bottom, the depth below the AUV of that contact point is entered into the Scan Depth array. The lighter shades on the array example indicate shallow depths while the darker shades indicate greater depths.

In Figure 3.3, the tilt angle is decreased to -60°, a 360° scan is performed and the depth data is entered in the array. Finally, the entire bottom scan is completed when the tilt angle is decreased to -90°, Figure 3.4. This is the bottom scanning and data storing process used in this thesis to build the Scan Depth array which, in turn, is used to

*Program code from [Ref. 21] was altered to create the elevation oblique 3D views in Figures 3.1-3.5.

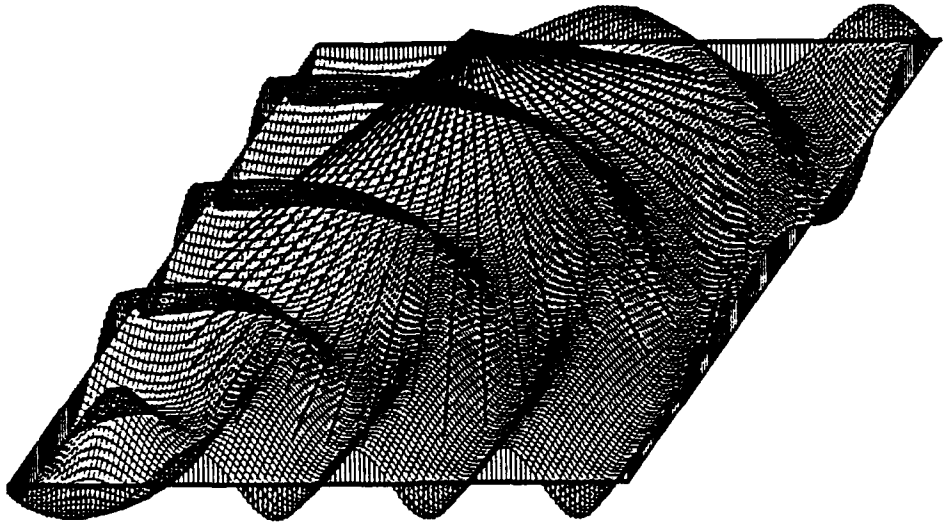


(a)

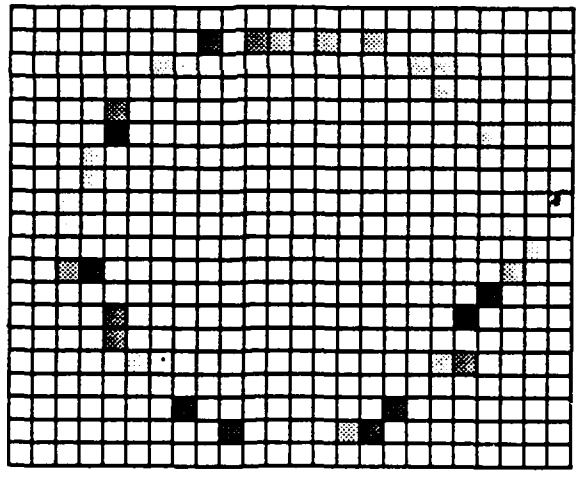


(b)

Figure 3.1
Zero Degree Tilt Scan (Sonar beam parallel to ocean surface)
a) Scan Illustration b) Corresponding Scan Depth Array

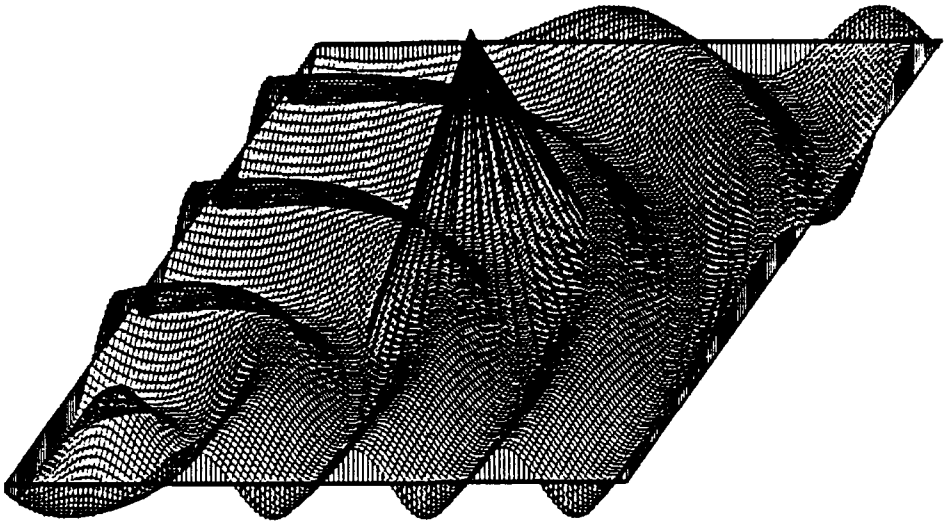


(a)

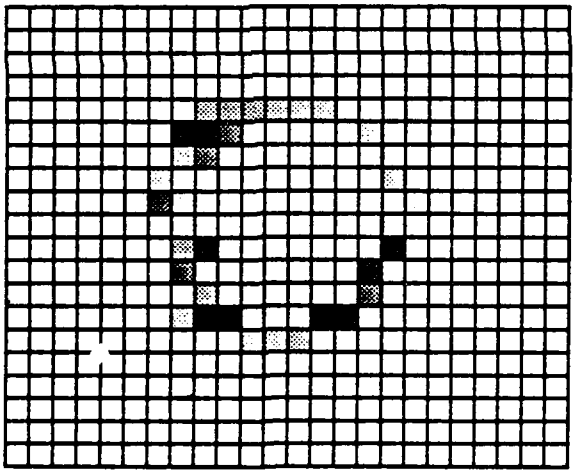


(b)

Figure 3.2
Thirty Degree Tilt Scan
a) Scan Illustration b) Corresponding Scan Depth Array

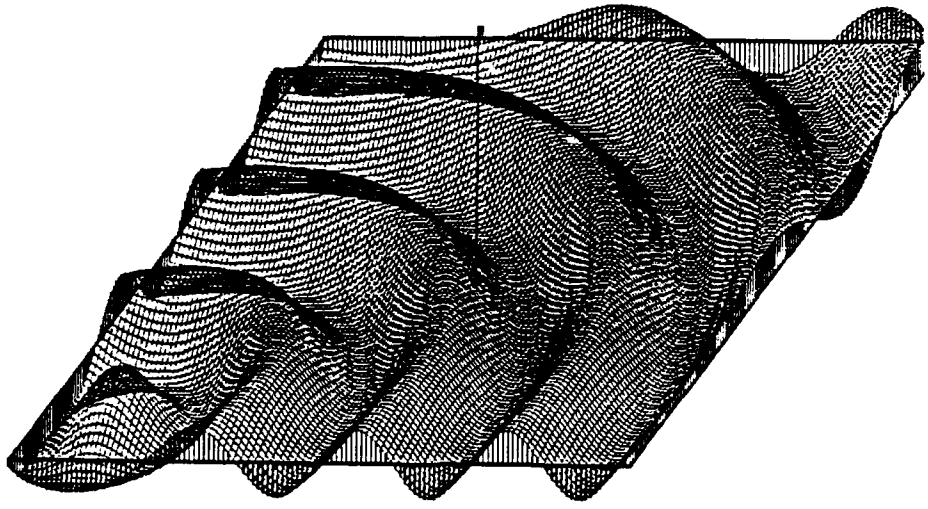


(a)

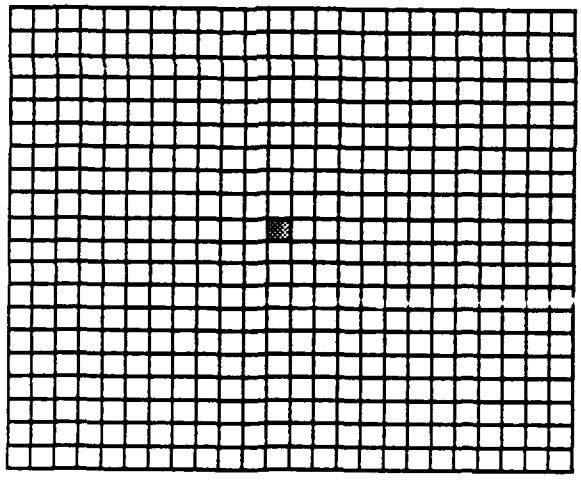


(b)

Figure 3.3
Sixty Degree Tilt Scan
a) Scan Illustration b) Corresponding Scan Depth Array



(a)



(b)

Figure 3.4
Ninety Degree Tilt Scan (Sonar beam perpendicular to ocean surface)
a) Scan Illustration b) Corresponding Scan Depth Array

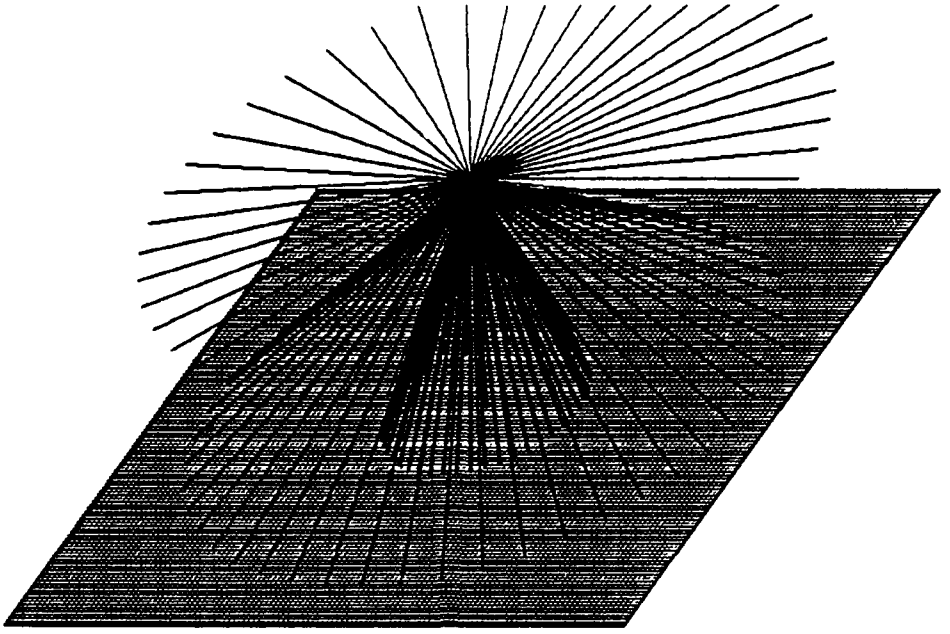
create a contour map of the ocean floor below the AUV. Figure 3.5 shows what the entire bottom scan process and the corresponding Scan Depth array look like. Obviously, the less the tilt angle is decreased after each 360° sweep, the more array cells will be filled with depth data and the closer the contour map, constructed from the array, will resemble the actual bottom terrain.

b. Contour Map Comparison

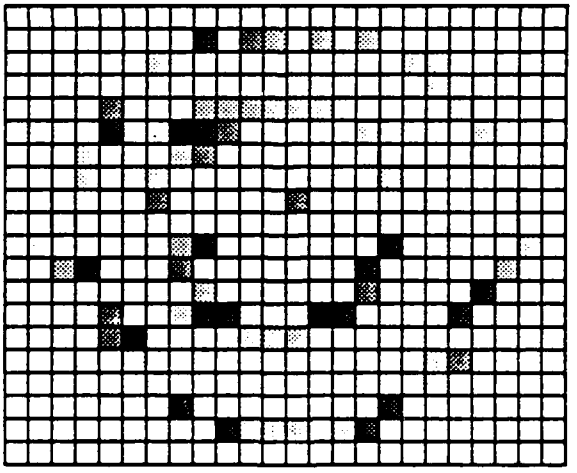
In order to detect AUV motion though, it is necessary to conduct two complete scans of the bottom. The first scan at the beginning of a time interval and the second at the end. If the contour maps are different, the AUV has moved. Figure 3.6a shows two simplified successive contour maps. The maps are different, so the AUV has moved, but in which direction and how far? If the first scan is laid over the second, as in Figure 3.6b, the scan differences become more apparent. The first or older scan, chronologically speaking, will be shifted over the bottom, or newer scan until the largest portions of both maps match. In this example, the most shallow parts of the first scan are in the upper right hand corner while the most shallow part of the second scan is directly in the center. The precise mathematics to determine which way to shift the top map to continually obtain a better match is discussed in more detail later in this chapter. This simplified example will try to match the shallow parts of the first and second maps. Figure 3.7a shows the first array shifted one cell to the left. The overlapping portions of the two maps match more closely than before, but still may match better. The second shift, Figure 3.7b, moves the first scan again, but down one cell this time. At this point the overlapping portions of the two scans match perfectly.

c. Direction and Distance Determination

When the two scans match, as in Figure 3.7b, the difference between the centers of the scans is used to determine the direction and magnitude of the AUV motion.

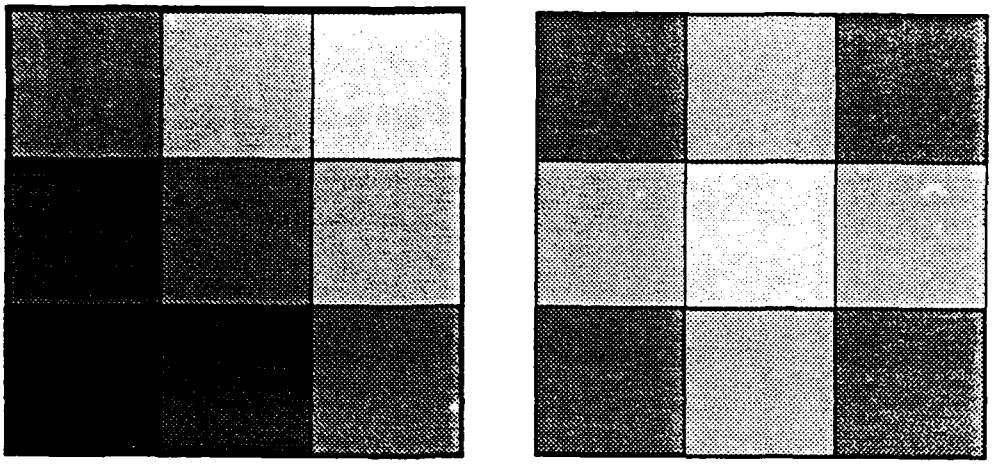


(a)



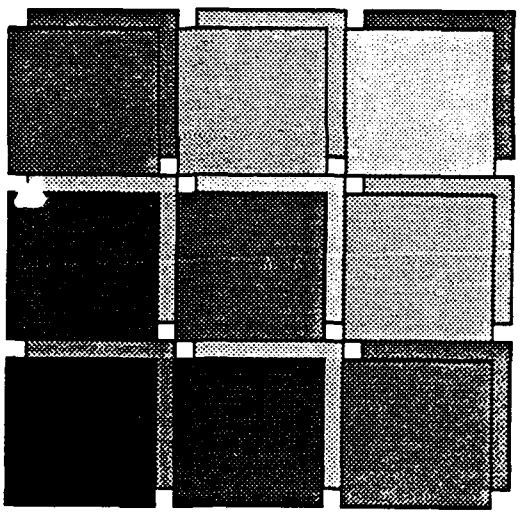
(b)

Figure 3.5
Complete Bottom Scan
a) Scan Illustration b) Corresponding Scan Depth Array



First Scan Second Scan

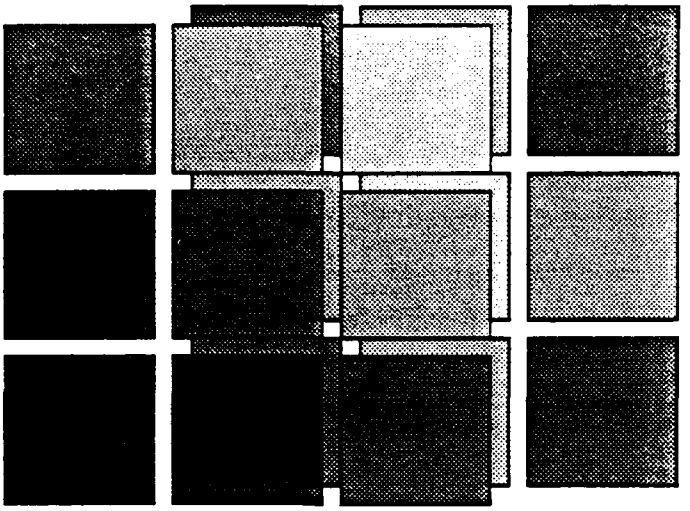
(a)



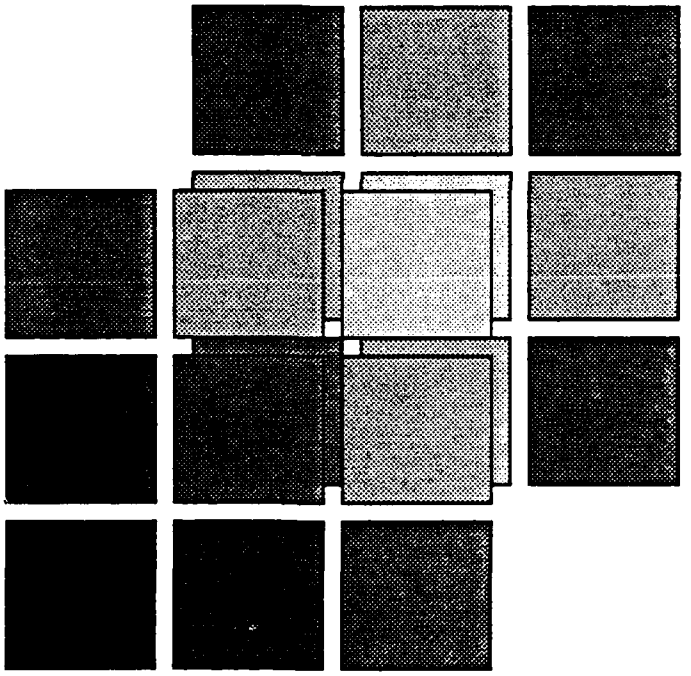
First Scan Over Second Scan

(b)

Figure 3.6
Consecutive Scan Depth Arrays
a) Side by Side b) Overlaid



(a)



(b)

Figure 3.7
Consecutive Scan Depth Array Comparison
a) First move b) Second move

The direction of AUV motion is the direction the bottom scan was moved relative to the top scan. In this example, the bottom scan moved in a direction of 045° from the top, so the AUV direction between scans was 045°. The distance traveled between scans is a hypotenuse length calculation using the horizontal cell and vertical cell shifts as the two perpendicular sides of the triangle. In this example, one left cell shift and one down cell shift were performed, making the distance between scans:

$$AUV\text{-}distance = \sqrt{(-1^2 \times -1^2) \times one\text{-}cell\text{-}distance} \tag{3.1}$$

where the one-cell-distance is determined by the sonar range setting. The larger the setting, the larger distance one cell move represents.

Describing the side by side scans in Figure 3.6a is a good double-check of these results. If the results are assumed correct, this description should verify the results. Assuming the AUV is traveling in a direction of 045°, toward the upper right hand corner of the scans, the first scan shows a deep section directly behind and a shallow section directly in front of the AUV (remember, the AUV is always in the center of the scans), or in descriptive terms, the AUV is leaving a deep area and approaching a shallow area, perhaps a sea mount. Obviously, the next scan should place the AUV closer to the sea mount. The second scan shows that the most shallow section is directly under the AUV, or the AUV is directly over the sea mount. The AUV has traveled from in front of the sea mount to directly over the sea mount between scans. (A diagonal distance of one cell.) This map comparison description supports the AUV motion results presented above.

C. WESMAR SS265 SONAR

1. System Description

The WESMAR SS265 OMNICOLOR Sonar, one of which is currently operational aboard the USCG Cutter Conifer home ported in Long Beach, California, contains very advanced and cost effective sonar and computer technology. This system allows the user to send a sweeping soundbeam in any direction from the vessel -- fore, aft, up, down and to either side. The WESMAR scanning sonar is often referred to as *underwater radar*. [Ref. 22]

The SS265 plots sonar targets on a color video display that exhibits high brightness and contrast, no fade, and no viewing parallax. Seven distinct colors are used to indicate target density: red represents maximum target echo strength, blue represents minimum target echo strength, and five other hues of the spectrum indicate targets of intermediate strength. Digital readouts on the display screen indicate the selected range, soundome tilt, hoist deployment, and the horizontal distance and depth to a user selected target. Any number of displays can be placed in the system, allowing remote viewing anywhere on the vessel. The different models made by WESMAR, "Western Marine Electronics," range from 6,000 to 50,000 dollars in price [Ref. 22]. Although the actual WESMAR sonar indicates target density, this simulation will assume all targets have the same density.

Some of the WESMAR's advanced features include a *self test, non-fading color display, power boost, sector scanning, soundbeam tilt control,* and a *stabilized soundome*. Each time the sonar is activated, a self-test examines the condition of the RAM and ROM computer memories, providing the user with an early warning of any potential trouble. The transducer scan motor and transducer tilt mechanism also undergo a complete automatic internal checkout before each use. [Ref. 20]

The video display is a full color look at the undersea that never fades from view. The computer memory stores each scan of the soundbeam - displaying bottom contours, wrecks, fish schools, reefs, pinnacles, channels and more, in nine brilliant colors. The RANGE RINGS make target tracking and interpretation simple. Each corner of the screen shows numeric readouts with instant information on target distance and depth, as well as soundbeam range and tilt. The transmit power of the WESMAR can be changed depending on depth situations. Low power is used for shallow water scanning to cut down on noise and false echos. High power, ten times low power, is used for deep water searching. [Ref. 20]

WESMAR's sector scanning system allows the user to monitor a full 360° display area or any portion from 15° to 330°. This allows the user to follow rapidly moving targets and scan small areas of particular interest. The WESMAR also allows the user to control the soundbeam angle. This control lets the user search anywhere in the water column - surface to bottom. A unique stabilization system of the WESMAR, corrects for most vessel pitch and roll during sonar scanning. The stabilization allows the sonar to perform effectively in varying seas. [Ref. 20]

2. Physical Environmental Data

There are four basic components of the SS265 sonar: the transducer (or soundome), which does the transmitting and receiving, housed in a hull-mounted sea chest, a hoist to raise and lower the transducer, a control panel, and a display unit (which can be either an everyday television receiver or a video monitor).

a. Control Console

The console is three inches high, 17.875 inches wide and is eleven inches deep. It weighs eight pounds. The console can operate in temperatures ranging from 32° to 122°F and in relative humidity no greater than 95 percent. [Ref. 20]

b. Transducer

The transducer (or soundome) is 5 and 3/4 inches in diameter, is 17.5 inches high and weighs 25 pounds (50 pounds with the hoist assembly). It can operate in temperatures between 0° and 130°F, inclusive. [Ref. 20]

3. Operational and Electrical Data

a. Control Console

The control console uses 12 to 15 volts DC to operate and has a negative ground. Its average current drain is 2.5 Amperes at 12 VDC and its average power consumption is 30 watts. The sonar ranges, 50, 100, 200, 300, 400, 600, 800, 1200, 1600 and 2400 feet or 15, 30, 60, 100, 150, 200, 300, 400, 600 and 800 meters can be selected at the console [Ref. 20]. This simulation uses the metric scale.

b. Transmitter

The transmitter frequency is 160 Khz, has 1000 watts (peak) in the HI Power mode and 100 watts (peak) in the LO Power mode and has a pulse length that varies with the range setting - .2 ms to 2.5 ms. The transmitter's beam-width is 6.5° [Ref. 20]. This simulation uses a beam-width of 8°.

c. Video Display

Any television receiver or monitor accepting U.S. Channels 3 or 4 (NTSC American television standard) or EIA RS 170 video monitor at 75Ω can be used as the video display device. Four ranges rings are displayed on the sonar return video on every range setting. [Ref. 20]

D. MATHEMATICAL MODEL OF SYSTEM

1. Terrain Sensing and Video Display

In order to model the sonar in this thesis, the sonar beam tip position must be known relative to the base (terrain origin). This beam tip position will be used to determine if and when the beam contacts the ocean floor. The beam tip position could be determined by solving a direct kinematics problem, similar to the method used in [Ref. 21] if the AUV and sonar system were to be modeled as a nine link manipulator. Nine orthogonal coordinate systems and Joint and Link parameters for each link could be established. Nine Denavit-Hartenberg transformation matrices (A - MATRICES) could be established to relate adjacent links, then these *A* matrices would produce the composite transformation matrix (T - MATRIX), which could then be used to calculate the beam tip location in base coordinates [Refs. 23, 24]. This extremely powerful method was considered overkill in this simulation. Instead, a much simplier Polar to Cartesian coordinate conversion process is used. Specifically, the *homogeneous-transformation matrix* used to calculate the AUV's position in base coordinates is

$$ {}^{0}A_{6} = \begin{bmatrix} c\Psi_a c\Theta_a & c\Psi_a s\Theta_a s\Phi_a - s\Psi_a c\Phi_a & c\Psi_a s\Theta_a c\Phi_a + s\Psi_a s\Phi_a & x_a \\ s\Psi_a c\Theta_a & c\Psi_a c\Phi_a + s\Psi_a s\Theta_a s\Phi_a & s\Psi_a s\Theta_a c\Phi_a - c\Psi_a s\Phi_a & y_a \\ -s\Theta_a & c\Theta_a c\Phi_a & c\Theta_a c\Phi_a & z_a \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2} $$

where c = cos, s = sin, Ψ_a = AUV Heading (Azimuth), Θ_a = AUV Dive Angle (Elevation) and Φ_a = AUV Roll Angle. The elements x_a, y_a and z_a in this matrix refer to the location of the center of the AUV sonar scan mechanism in Earth coordinates.

Given Equation (3.2), from Figure 3.8, the transformation from AUV coordinates to beam tip (end effector) coordinates is evidently given by

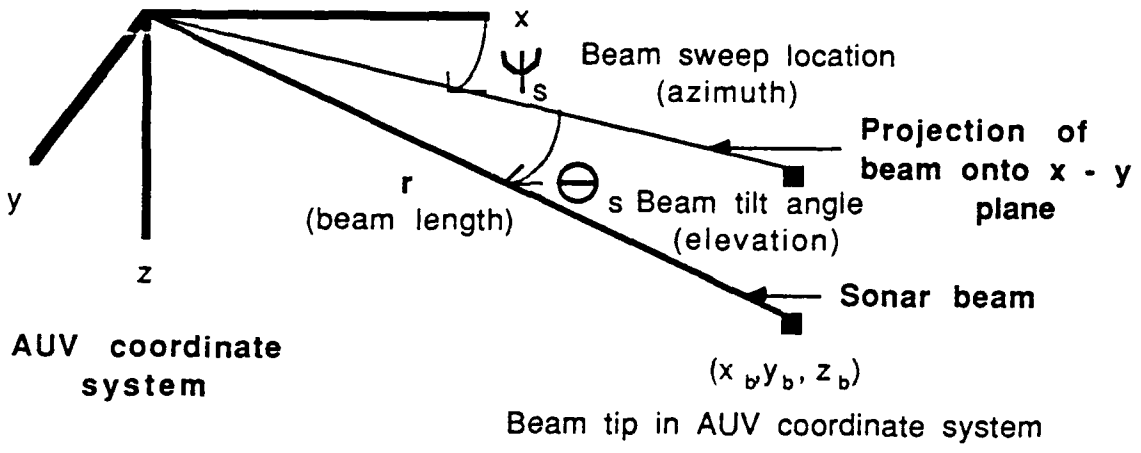


Figure 3.8
Sonar Beam Tip Polar to Cartesian Transform Illustration

$$ {}^{6}A_{9} = \begin{bmatrix} beam\text{-}length \times \cos\Theta_b \times \cos\Psi_b \\ beam\text{-}length \times \cos\Theta_b \times \sin\Psi_b \\ -beam\text{-}length \times \sin\Theta_b \\ 1 \end{bmatrix} \tag{3.3} $$

where Θ_b = Tilt angle and Ψ_b = Sweep location. From Equations 3.2 and 3.3, the matrix ${}^{0}A_{9}$ can be calculated as [Ref. 21]

$$ {}^{0}A_{9} = {}^{0}A_{6}\,{}^{6}A_{9} \, . \tag{3.4} $$

The upper right 3 x 1 submatrix of this composite matrix represents the location of the sonar beam tip in base coordinates. This submatrix is used to generate the following equations to calculate the beam tip location in base coordinates:

$$x\text{-}coord_b = x\text{-}coord_a + (\cos\Psi_a \cos\Theta_a\, beam\text{-}length \cos\Theta_b \cos\Psi_b) + ((beam\text{-}length \cos\Theta_b \sin\Psi_b) \times ((\cos\Psi_a \sin\Theta_a \sin\Phi_a) - (\sin\Psi_a \cos\Phi_a))) - ((beam\text{-}length \sin\Theta_b) \times ((\cos\Psi_a \sin\Theta_a \cos\Phi_a) + (\sin\Psi_a \sin\Phi_a))) \quad (3.5)$$

$$y\text{-}coord_b = y\text{-}coord_a + (\sin\Psi_a \cos\Theta_a\, beam\text{-}length \cos\Theta_b \cos\Psi_b) + ((beam\text{-}length \cos\Theta_b \sin\Psi_b) \times ((\cos\Psi_a \cos\Phi_a) + (\sin\Psi_a \sin\Theta_a \cos\Phi_a))) - ((beam\text{-}length \sin\Theta_b) \times ((\sin\Psi_a \sin\Theta_a \cos\Phi_a) - (\cos\Psi_a \sin\Phi_a))) \quad (3.6)$$

$$z\text{-}coord_b = z\text{-}coord_a - (\sin\Theta_a\, beam\text{-}length \cos\Theta_b \cos\Psi_b) + (\cos\Theta_a \sin\Phi_a\, beam\text{-}length \cos\Theta_b \sin\Psi_b) - (\cos\Theta_a \cos\Phi_a\, beam\text{-}length \sin\Theta_a) \; . \quad (3.7)$$

These beam position equations are used in the bottom-scanning model. Figure 3.9 is a flow chart of the entire simulation program while the Appendix contains the commented computer program code. Figures 3.10 and 3.11 are flowcharts of the terrain scanning process. In scanning the terrain, the sonar's beam length is increased in amounts equal to 1/15 of the sonar range setting. After each increase of the beam, Equations 3.5 - 3.7 are used to calculate the **x**, **y** and **z** coordinates of the end of the beam. The calculated value of **z** is then compared to the value of the ocean depth of the actual terrain stored at the **x** and **y** coordinate of the beam tip. If the beam tip depth is less than the actual terrain depth, the beam tip is above the ocean floor at that point. The beam length is then increased and the process starts again. If the beam tip depth is greater than or equal to the actual depth, the final sonar beam length is calculated by Equation 3.8.

$$beam\text{-}length_{FINAL} = beam\text{-}length - \frac{beam\text{-}increment}{2} \quad (3.8)$$

The half beam increment adjustment brings the beam length back to a value closer to the actual range.

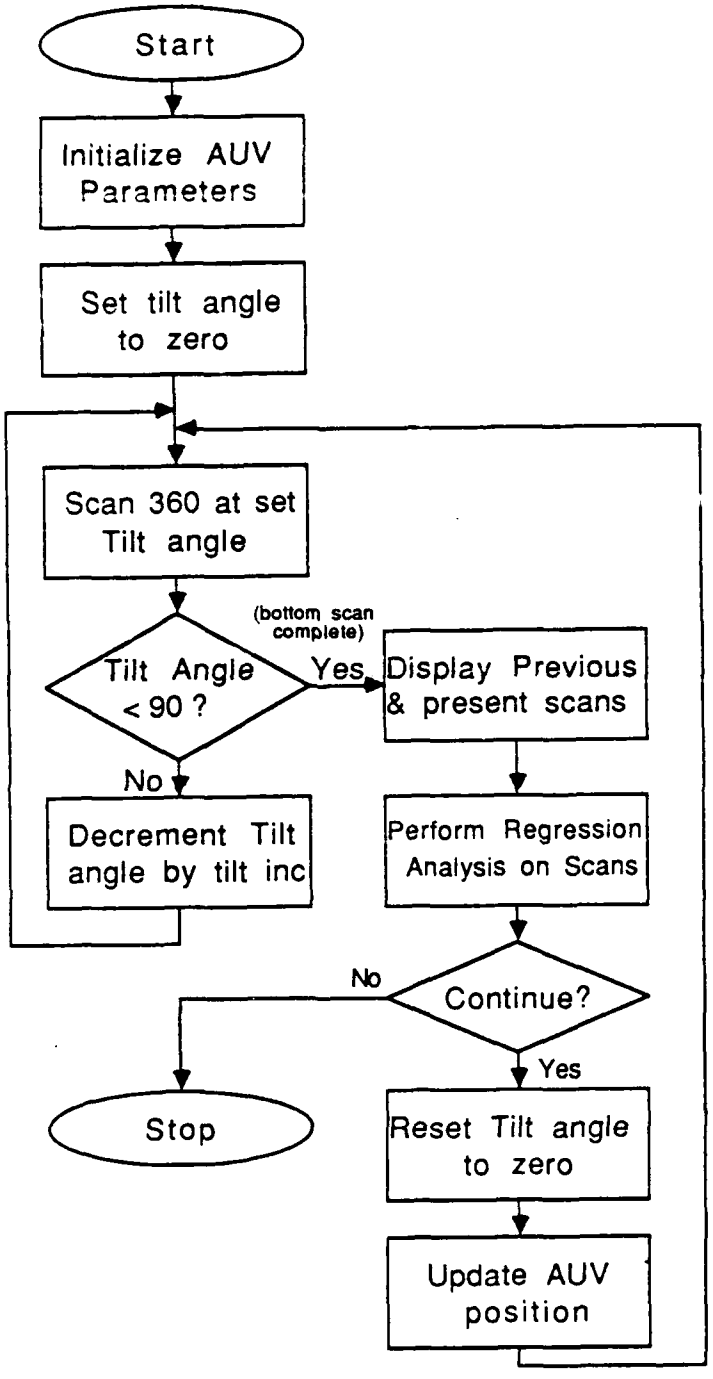


Figure 3.9
Overall System Flowchart

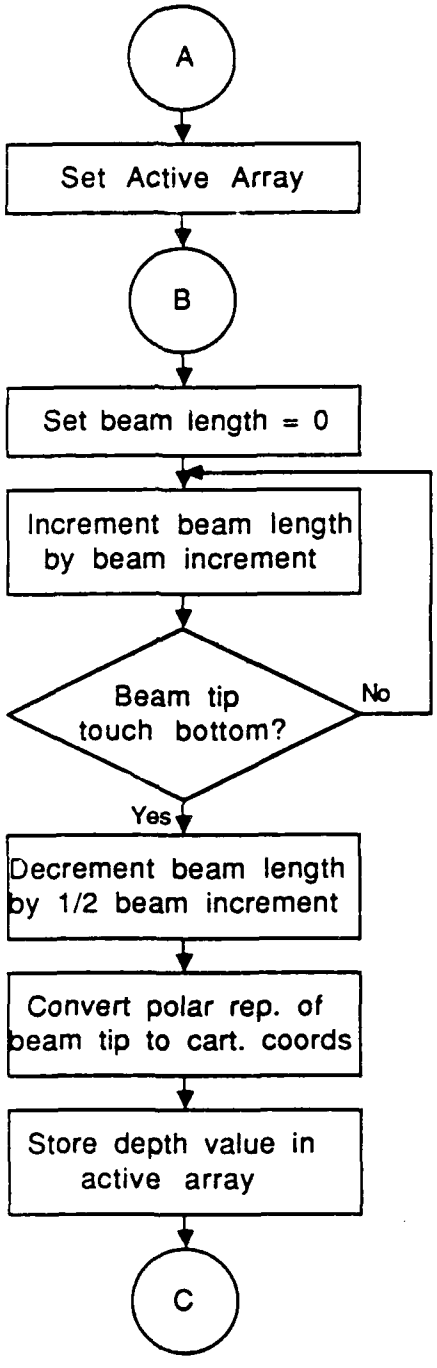


Figure 3.10
Terrain Scanning Flowchart (part 1 of 2)

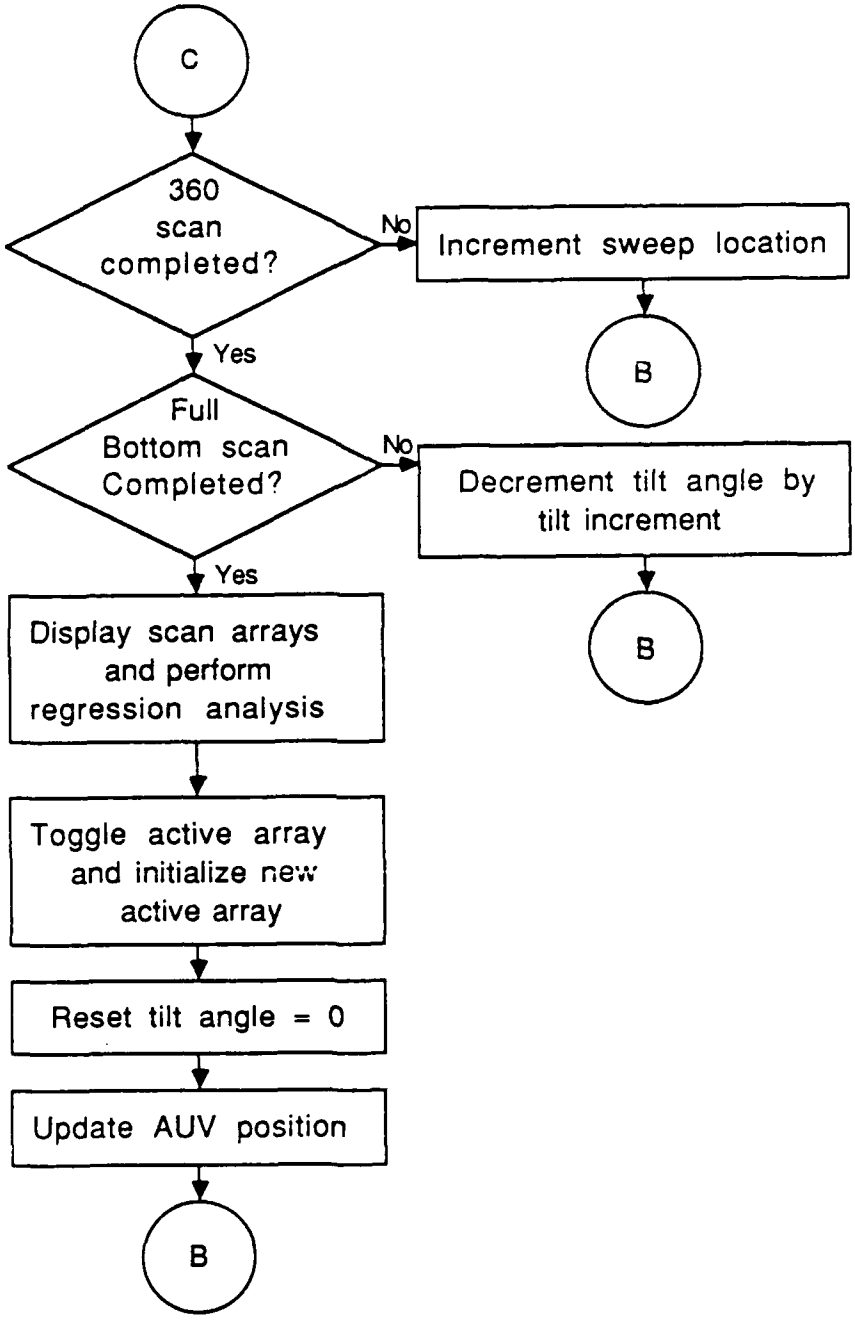


Figure 3.11
Terrain Scanning Flowchart (part 2 of 2)

To start the scanning process, the sonar tilt angle, the beam length and the sweep location are all set to zero. The beam is lengthened until it reaches its maximum range, determined by the sonar range setting, or until it contacts the ocean floor. The sweep location is increased by eight degrees, the simulated beam width, and the beam is sent out from the AUV again. When the sweep location returns to 360°, the tilt angle is decreased by the tilt increment (determined by the user), the sweep location and beam length are reset to zero and the sweep process is started again. When the tilt angle becomes less than -90°, a full bottom scan has been completed.

Whenever the beam reaches maximum length or contacts the bottom, the sonar video screen is updated. Figure 3.12 shows that the video screen is separated into 45 eight degree arcs and each arc consists of 15 polygons. These fifteen polygons correspond to the 15 beam increments, so the edge of the sonar screen is the sonar's maximum range, regardless of the range setting. The sweep location determines which arc will be updated

$$arc-number = \frac{sweep-location}{8} \tag{3.9}$$

and the beam length determines which polygon will be filled red (bottom contact color).

$$polygon-fill-number = \frac{beam-length}{beam-increment} \tag{3.10}$$

All polygons inside of the contact polygon are colored blue and all polygons outside of the contact polygon are colored cyan, indicating sonar shadow. If no bottom contact is made, the entire arc is colored blue.

2. Sonar Return Data Storage and Regression Analysis

When the sonar beam contacts the bottom, the Polar coordinate representation of the beam tip in sweep location, tilt angle and beam length is converted to AUV

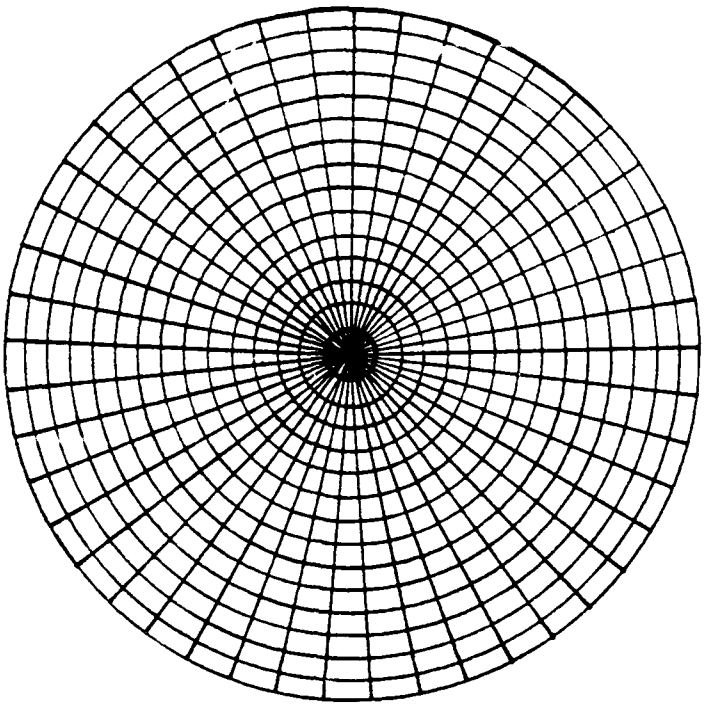


Figure 3.12
Sonar Video Screen Breakdown
45 Eight Degree Arcs with 15 Polygons in each Arc

relative Cartesian coordinates. To simplify all following calculations, it is assured that the AUV is pointing north with zero pitch and roll angles. Thus, the location of the sea bottom in AUV body coordinates is given by

$$x_{beam-tip} = \frac{beam-length}{beam-increment} \times cos(tilt-angle) \times cos(sweep-location) \tag{3.11}$$

$$y_{beam-tip} = \frac{beam-length}{beam-increment} \times cos(tilt-angle) \times sin(sweep-location) \tag{3.12}$$

$$z_{beam-tip} = \frac{beam-length}{beam-increment} \times sin(tilt-angle) \tag{3.13}$$

This information generates the Scan Depth array which can be converted to a contour map of the ocean floor. The Scan Depth array shown in Figure 3.13 is a 30 x 30 cell array which retains the sonar return information. The center of the array corresponds to the AUV position. Equations 3.11 and 3.12 are used to determine the array indices

$$x-index = 15 - x_{beam-tip} \tag{3.14}$$

$$y-index = 15 + y_{beam-tip} \tag{3.15}$$

at which the $z_{beam-tip}$ or depth below AUV, value is stored.

At each completion of a bottom scan, the previous and current arrays are compared using the *Grid Search* method [Ref. 21]. In this approach, there is a criterion function, Φ, which is the sum squared error between the depth values of the overlapping cells of the successive arrays. Both overlapping cells must contain depth values to be used in the summation. Specifically,

$$\Phi = \frac{1}{n}\sum(D_1(x_i, y_i) - D_2(x_i + \Delta x, y_i + \Delta y))^2 \tag{3.16}$$

where n is the number of overlapping cells where both contain data, D_1 is the old scan array and D_2 is the new scan array, and $(\Delta x, \Delta y)$ is the unknown movement of the AUV between two successive bottom scans.

A flowchart of the Grid Search process is shown in Figure 3.14. To start the analysis, the old scan is *placed directly over* the new scan and the starting criterion function for the center cell, Φ_0, is calculated. Using the procedure described in [Ref. 25], the 3 x 3 array cell mask is set up with the cell of interest at the point where $\Delta x = 0$ and $\Delta y = 0$, with its associated value of Φ_0. Table 3.1 shows the relative distances of the cells and the criterion functions associated with them. The top array is shifted so that the top

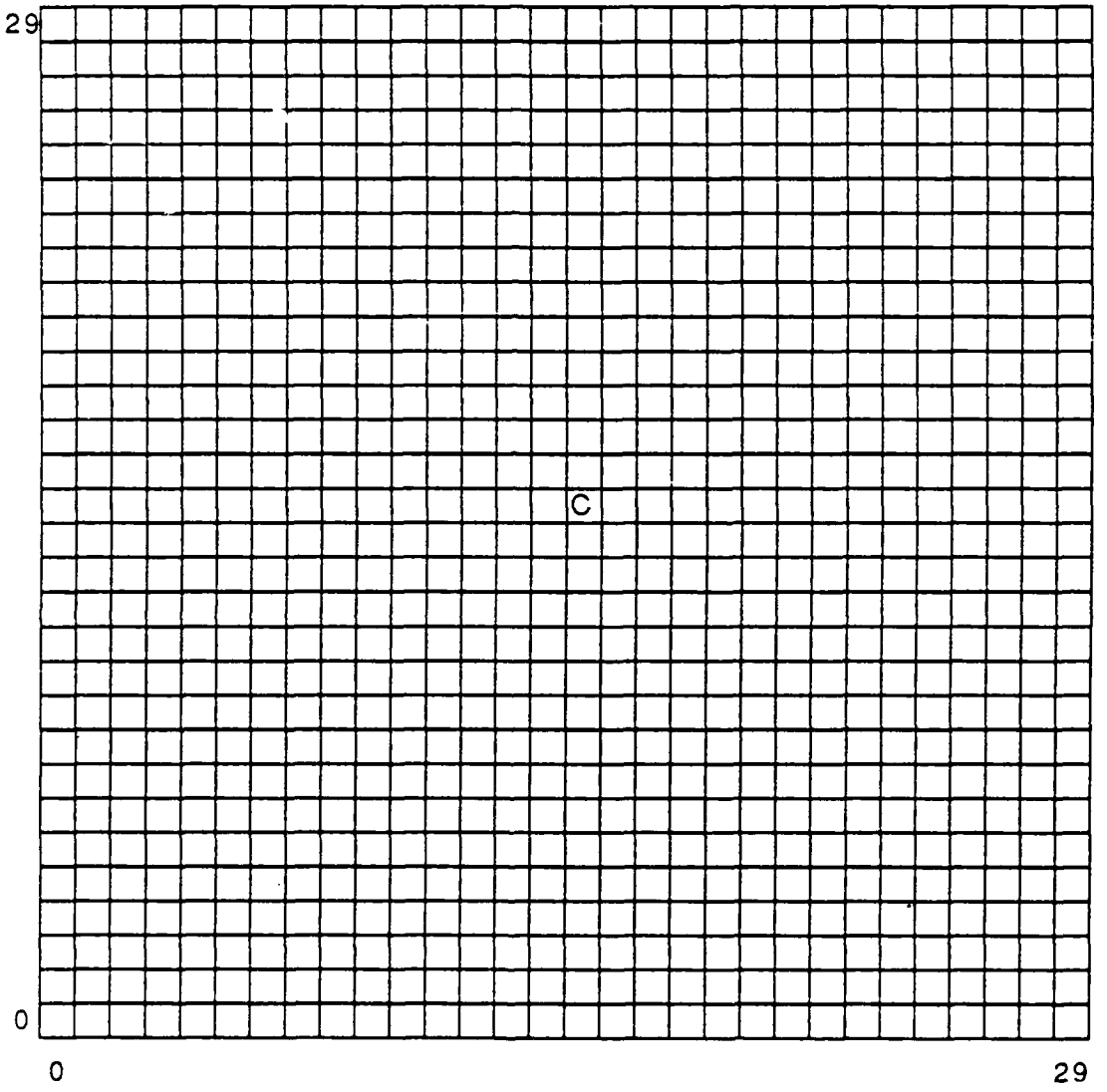


Figure 3.13
Scan Data Storage Array

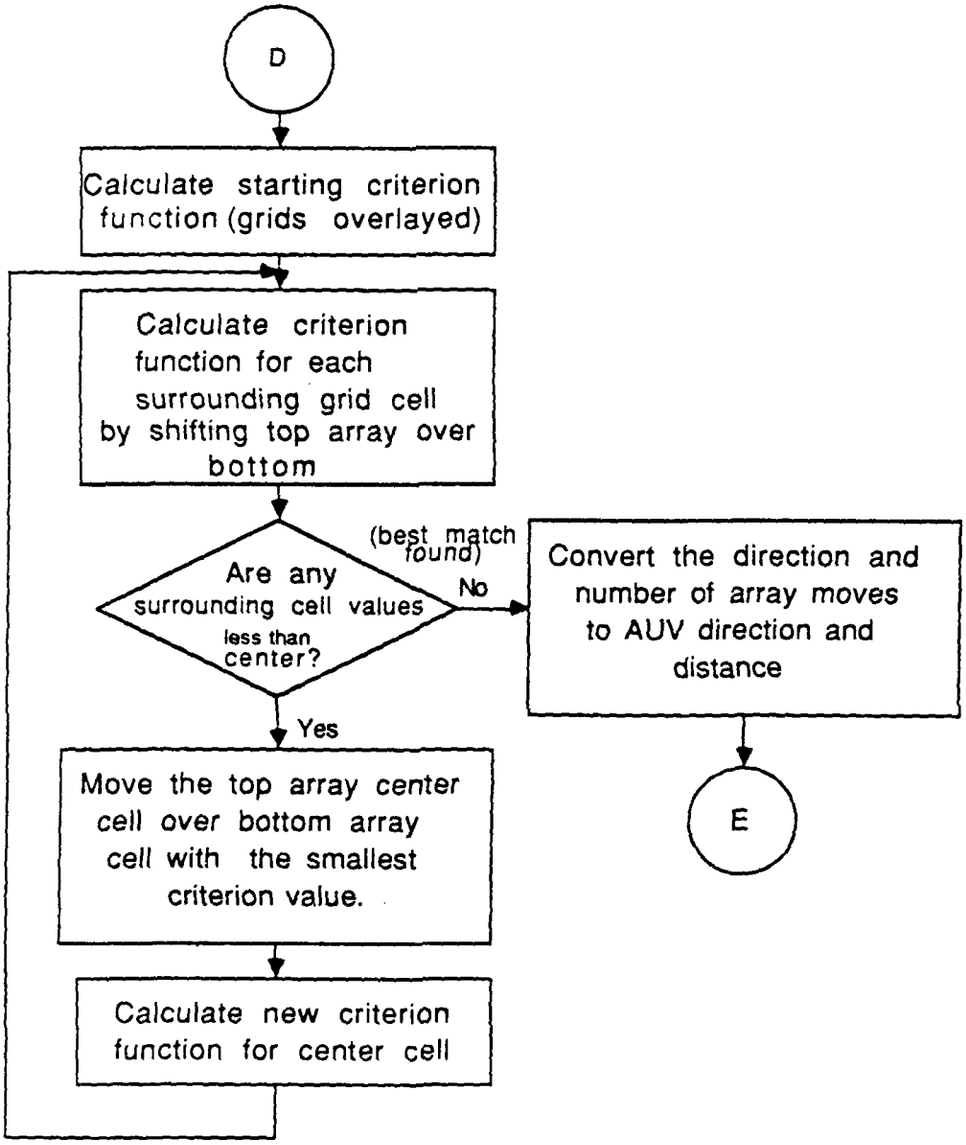


Figure 3.14
Grid Search Flowchart

array center cell is placed over each cell surrounding the bottom array center cell in the order noted in Table 3.1, and the corresponding Φ's are calculated. The top array is then

TABLE 3.1
RELATIVE DISTANCES OF TERRAIN
CELL MASK AND CRITERION FUNCTIONS

$\Delta x = -1$ $\Delta y = 1$ Φ_6	$\Delta x = 0$ $\Delta y = 1$ Φ_7	$\Delta x = 1$ $\Delta y = 1$ Φ_8
$\Delta x = -1$ $\Delta y = 0$ Φ_4	$\Delta x = 0$ $\Delta y = 0$ Φ_0	$\Delta x = 1$ $\Delta y = 0$ Φ_5
$\Delta x = -1$ $\Delta y = -1$ Φ_1	$\Delta x = 0$ $\Delta y = -1$ Φ_2	$\Delta x = 1$ $\Delta y = -1$ Φ_3

moved so that the top array center cell is over the least value Φ and the bottom array cell with that least value Φ becomes the new cell of interest. The associated Φ's are again calculated as described earlier. Once the center cell criterion function, Φ_0, contains the least value of all surrounding Φ's, the best match of the two successive scans has been found.

E. SIMULATION FACILITIES

1. Hardware and Overall System Description

The Naval Postgraduate Graphics Laboratory has three IRIS (Integrated Raster Imaging System) graphics workstations. These workstations are high-performance, high-resolution color computing systems for 2-D and 3-D computer graphics. They

provide a powerful set of graphics primitives in a combination of custom VLSI circuits, conventional hardware, firmware and software. [Refs. 26-29]

The IRIS 2400 turbo workstation, the model this simulation was designed, written and tested on, consists of two 72 MB Winchester disk drives, a high-resolution (1024 x 768) 19-inch RGB color monitor, an 83-key up-down encoded keyboard, a three-button mouse, an external dial box with eight dials, a Digital VT220 side terminal, 32 pixel color bitplanes and a hardware matrix multiplier Geometry Pipeline. This IRIS model has a Motorola 68020 CPU which can perform two million instructions per second. In graphics terminology, this equates to 80,000 transformations per second or 1000 Z-buffered, Gouraud shaded polygons per second.

The interesting feature about the IRIS graphics system is that it implements many of its graphics capabilities and functions in hardware. This allows the IRIS to perform 3D color graphic simulations much faster. Drawing polygons, filling polygons, backface polygon removal, coordinate transformations, plane clipping and shading are all computationally intensive functions that other graphics systems perform with software and pay a high penalty on performance as a result. All are done within hardware on the IRIS.

The three main hardware pipeline components in the IRIS system are the applications/graphics processor, the Geometry Pipeline and the raster subsystem. Each of the extensive selection of graphics commands, which significantly increase programmer productivity, is first processed by the applications/graphics processor. These processed commands are then sent through the Geometry Pipeline, which performs matrix transformations, multiplications and coordinate normalization, scaling and transformations. The raster subsystem accepts the Geometry Pipeline output as input and

performs low level drawing, filling, shading, depth-cueing and hidden surface removal functions. [Ref. 26]

2. Programming Language

Although the numerous IRIS graphics functions may be called by many programming languages such as Pascal and Fortran, the language best suited to build a simulation on the IRIS is *C*. The *C* programming language maintains compatibility with the IRIS system software, which is also written in *C*, and is also well suited for the IRIS's Unix4.2BSD operating system.

3. Graphical Objects

The IRIS graphics system provides the ability to group together graphics commands relating to a single object. This allows the group of commands to be treated and manipulated as a single graphical *object*. The *object* can be rotated, scaled, transformed or even combined with other *objects* to form more complex *objects* or displays.

The *object* itself can also be edited and redisplayed very rapidly by deleting, editing, or adding primitive drawing commands in the *object* group. This editing capability removes the necessity of retyping every primitive drawing command in the *object* for redisplay, even if only a small change is desired.

4. Double Buffering

The IRIS system screen image is stored in the hardware bitplanes. There are 32 bitplanes and each bitplane provides one bit of storage per pixel (picture element). The color and brightness of each pixel is determined by the RGB (Red Green Blue) value stored in the pixel location of the bitplanes.

These bitplanes can be used in two different modes, *single* or *double buffer*. In single buffer mode, the image on the screen is updated as the drawing functions are

performed in the program. In double buffer mode, half of the bitplanes are used as the *front buffer* and the other half are used as the *back buffer*. This allows the image to be drawn in the back buffer, while the front buffer is being displayed, and then swapped with the front buffer when the image is completed. This ensures that no incomplete or changing image will be displayed. This simulation uses the *double buffer* mode to perform the sonar display animation.

F. SUMMARY

This chapter has presented a detailed description of the sonar system being simulated, the mathematical models needed to perform terrain scanning and regression analysis, and finally a description of the simulation facilities. The next chapter will present the model used to simulated the ocean floor terrain and a description of the complete simulation program, which incorporates the mathematical models derived in this chapter.

IV. COMPUTER SIMULATION MODEL

A. INTRODUCTION

In Chapter III, the AUV motion detection problem and the mathematics of terrain sensing, data storage, and regression analysis were discussed. In order for these mathematical schemes to be tested, it is necessary to make use of either an actual vessel (surface or subsurface), with a WESMAR sonar on board, or an accurate computer simulation. The choice for this thesis was obviously the computer, specifically the IRIS graphics system described in the previous chapter. In this chapter, the computer system model is presented that will allow sonar and AUV simulation using the mathematical models presented earlier.

The model used to simulate the terrain is discussed, along with a description of each module in the computer simulation program. A users manual, with instructions on how to operate the simulation program, is also presented in this chapter.

B. TERRAIN MODEL

The Naval Postgraduate School Computer Science department maintains an actual terrain database for computer terrain simulation. This data was obtained from the U.S. Army installation, *Fort Hunter-Liggett,* in central California. The terrain data area is 36 by 35 kilometers and elevation and vegetation values are taken each 12.5 meters. The terrain model used in this thesis is a one kilometer square grid in a location with wide ranging elevations. The bottom data array size is 80 by 80, allowing for 6400 elevation values. Each array index represents a distance of 12.5 meters.

The bottom data array is created by loading the 6400 elevation values into the array. These elevation values are then treated as *depth* values. The high elevation numbers

represent deep depth values while low elevation values signify shallow points. The ocean floor terrain simulation is basically the *Fort Hunter-Liggett* one kilometer square grid inverted.

The ocean floor contour chart displayed in the initialization and scanning portions of the program is created by converting the depth values to corresponding colors. The deepest point in the bottom database is converted to black, the most shallow point is converted to white and all points in-between are converted to an appropriate shade of blue (light for shallow, dark for deep).

C. MODULE DESCRIPTION

1. Data Storage and Regression Analysis Modules

Four modules perform sonar return data storage and regression analysis functions. The first is **Disp_arrays.c** which displays two successive terrain scans by converting the depth value in the scan data storage arrays to a corresponding color and then compares the two scans by performing a gird search regression analysis technique to determine AUV motion. **Init_arrays.c** is the procedure that initializes the two successive terrain scan data storage arrays. **Store_data.c** converts the sonar return data from polar coordinates, sonar-beam azimuth, tilt and length, to cartesian coordinates, x-coord, y-coord and depth. The x-coord and y-coord values are converted into scan data storage array indices where the depth value is then stored. Store_data.c stores data only in the active scan array. **Up_arrays.c** keeps track of which scan data storage array is active, resets the toggle when a scan is completed, and reinitializes the new active array before new scan data is stored.

2. Graphical Object Modules

Twenty-two modules are dedicated to generating and editing the graphical objects in this simulator. The first, **Draw_numbers.c**, is a procedure that creates large

numbers, used for numerical readout, that are displayed on the video portion of the sonar. **Edit_toggles.c** is a procedure that displays the toggle configuration, up or down, on the sonar panel, depending upon the toggle's current setting. **Make_arrows.c** creates the direction and velocity arrows for the AUV course and speed, AUV course over ground, and speed over ground and the ocean current set and drift. The contour map of the ocean floor is created by **Make_chart.c**. **Make_depth.c** builds the depth indicator bar which is used when setting initial AUV parameters. **Make_dials.c** builds the dials on the sonar control panel. These dials are then changed by **Edit_dials.c** whenever the dials on the dial box are rotated. The instructions on the AUV parameter initialization screen are made by **Make_inst.c**. **Make_mask.c** makes the sonar video scan mask. This scan mask is used to reduce the sonar scan video from its default setting of 360°. The scan mask is updated by **Edit_mask.c** whenever the sector dial indicates a mask change. The four range rings that overlay the sonar video screen are created by **Make_over.c**. **Make_panel.c** builds the sonar control panel. The AUV position indicator *plus* symbol used on the ocean bottom contour map during the AUV parameter initialization is made by **Make_plus.c**. **Make_read.c** makes the AUV instrument panel used to set initial AUV parameters and then to change AUV parameters, during scanning. The movable range ring, made by **Make_ring.c**, is used to measure contact distance and depth from the AUV. **Make_rtn.c** creates the actual sonar video return, forty-five eight degree arcs, that appear on the video screen. The scan heading line, made by **Make_scline.c**, indicates the zero degree azimuth of the sonar beam. **Edit_scline.c** is the procedure that edits the scan heading line whenever the scan heading direction toggle is utilized. **Make_sweep.c** builds the sonar sweep line indicating the present sonar beam azimuth angle. The sonar sweep line is updated by **Edit_sweep.c**. The tilt angle indicator in the upper left hand corner of the sonar video is generated by **Make_tilt.c**. This tilt angle indicator displays the *elevation* angle of the sonar beam. Editing of the tilt indicator is

handled in **Edit_tilt.c**. **Make_video.c** builds the video portion of the sonar simulator and also contains the mathematical machinery to track the sonar beam, which is necessary for terrain scanning.

3. Simulation Control Modules

The first of four modules used in real time simulation control is **Get_sets.c**. This procedure places the simulation back in the AUV parameter initialization mode where **Read_sets.c** converts the mouse and dial readings into AUV parameter settings. **Read_ctrls.c** converts the mouse and dial readings into AUV parameter settings, during sonar scanning. Allowing for AUV control in real time. In order to determine the completion of a terrain scan, **Scan_ctrl.c** calculates which of the three scan modes the simulation is in and then, depending on the mode, determines if a scan has been completed.

4. Miscellaneous Modules

Get_posit.c calculates the next position of the AUV depending on AUV course, speed and dive angle and the ocean current direction and magnitude. The procedure that performs all the graphics package initialization is **Init_iris.c**. This procedure contains the workstation initialization primitives, cursor description, color generation and more. **Load_data.c** reads elevation data from the *Fort Hunter-Liggett* terrain file into the ocean floor depth array. At the end of each simulation loop, all boolean flags are reset to *FALSE* by **Reset_flags.c**. The final module is **Main_sonar.c**. This module is the main program driver that calls all other modules.

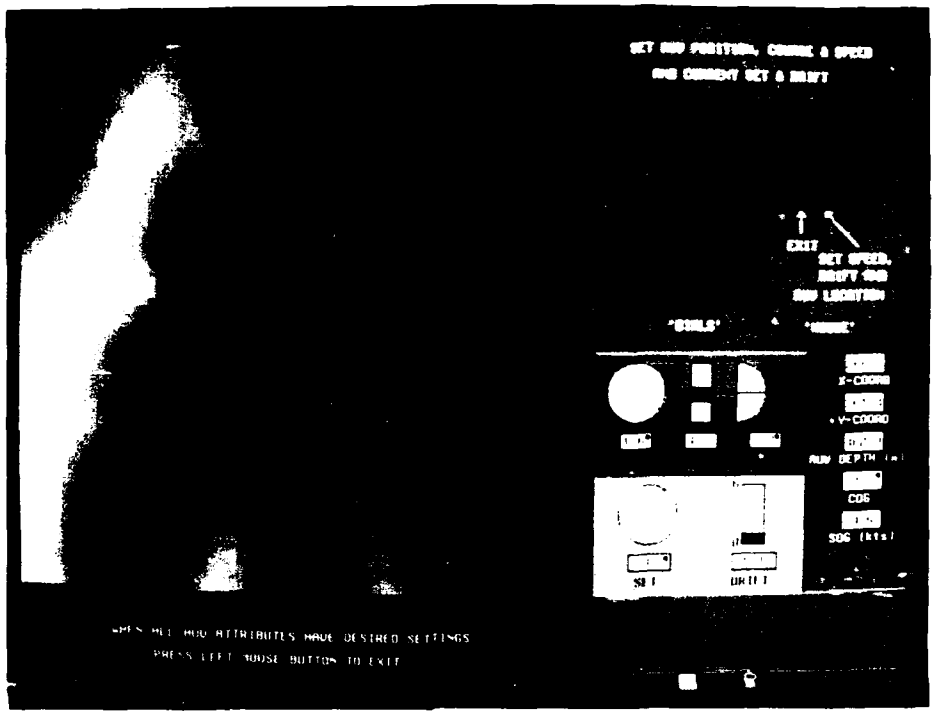
D. USER'S MANUAL

The WESMAR sonar simulator consists of three major parts: Initialization, Scanning/Storing, and Scan Storage Array Display. The initialization screen is displayed

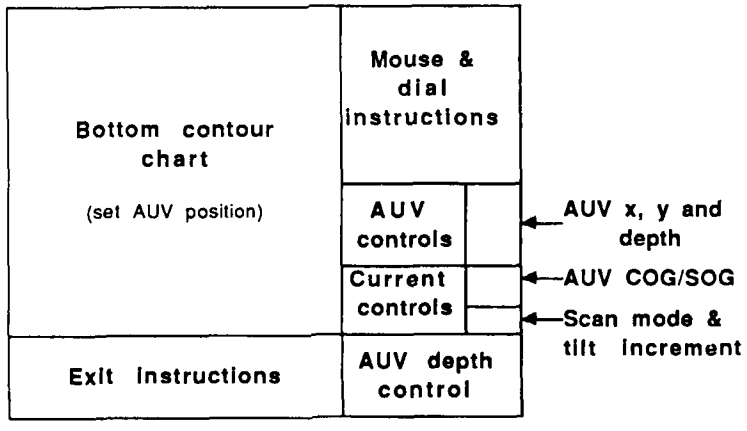
in Figure 4.1, the scanning screen in Figure 4.2 and the scan storage array display in Figure 4.3. The entire simulation program may be controlled using these three screens.

The WESMAR sonar simulator may be started by entering *wesmar* on the IRIS graphics workstation side terminal. Since all regression analysis results are displayed on the screen in which the program was started, *wesmar* must be entered on the side terminal to avoid graphic display interference. First to appear on the graphics terminal will be the AUV parameter initialization screen. At this point, initial AUV chart location, course, speed, dive angle and be set along with ocean current direction and magnitude.

The initial position of the AUV may be set by placing the cursor at the desired location over the ocean floor contour chart and pressing the middle mouse button. The position indicator will move to the location of the cursor. The initial depth of the AUV may be adjusted by placing the cursor over the depth indication bar in the lower right hand corner of the screen and pressing the middle mouse button. The depth bar symbolizes the depth of the water at the current AUV location, so to set the initial AUV depth at half way to the bottom, place the cursor in the middle of the bar. The AUV speed and the ocean current magnitude can be adjusted in an identical fashion by placing the cursor over the AUV speed or ocean current drift bar and pressing the middle button. By turning the appropriate dials on the dial box, the AUV course, dive angle and ocean current direction may be set. The scanning mode may be set at this point also. This simulator has three modes: just scanning with no sonar return data being stored, one 360° scan with the return data being stored, and a complete hemispherical scan with the return data being stored. Select the scan mode by pressing the 1, 2, or 3 key respectively. If mode 3 is selected, press the up arrow key until the tilt increment reaches the desired value. The tilt increment determines the number of degrees that the sonar beam elevation

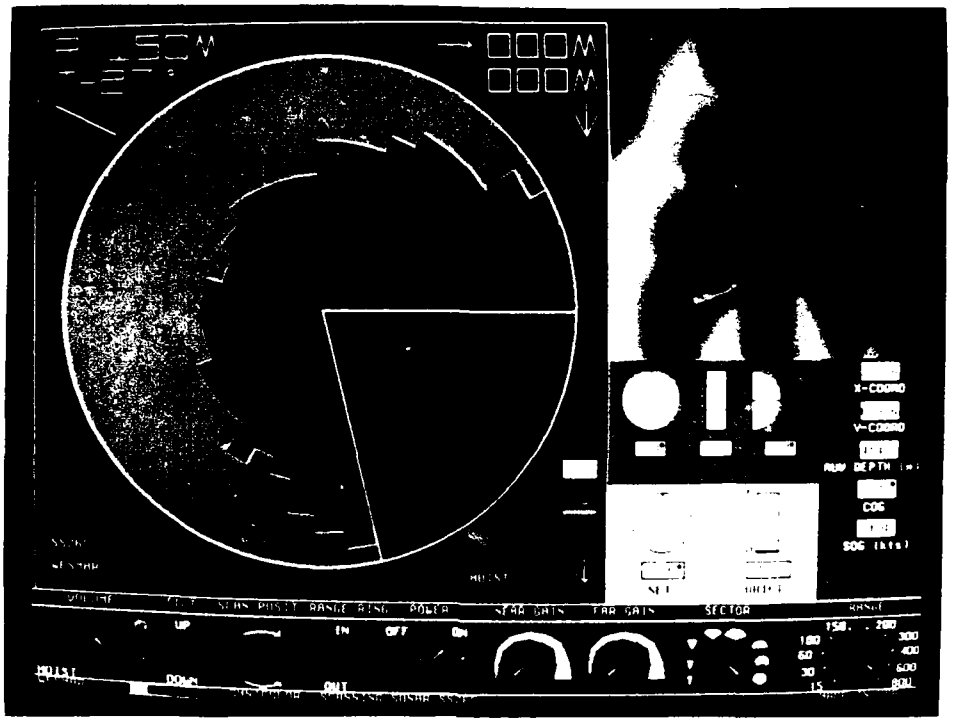


(a)

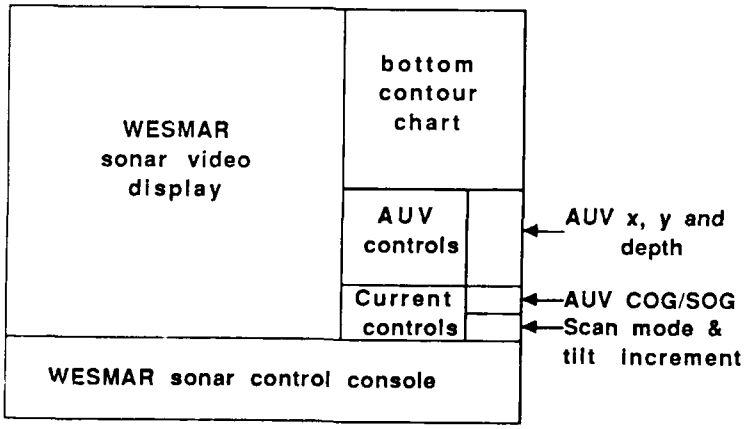


(b)

Figure 4.1
Initialization Screen
a) Graphics Screen b) Screen Description

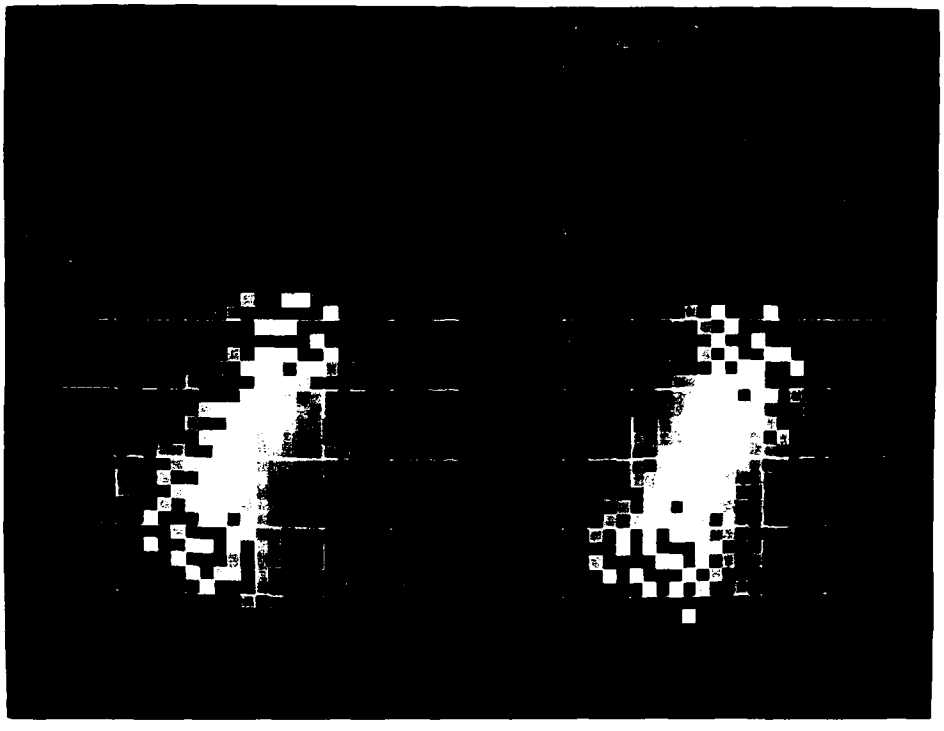


(a)

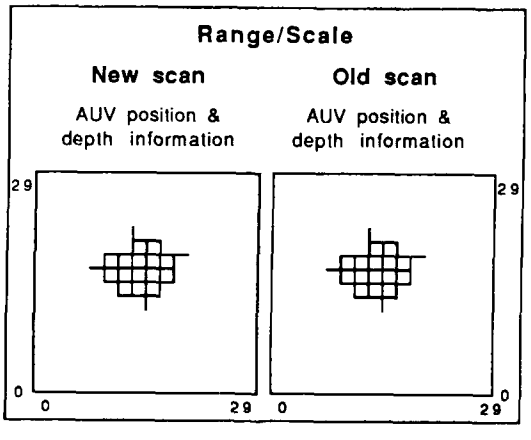


(b)

Figure 4.2
Scanning Screen
a) Graphics Screen b) Screen Description



(a)



(b)

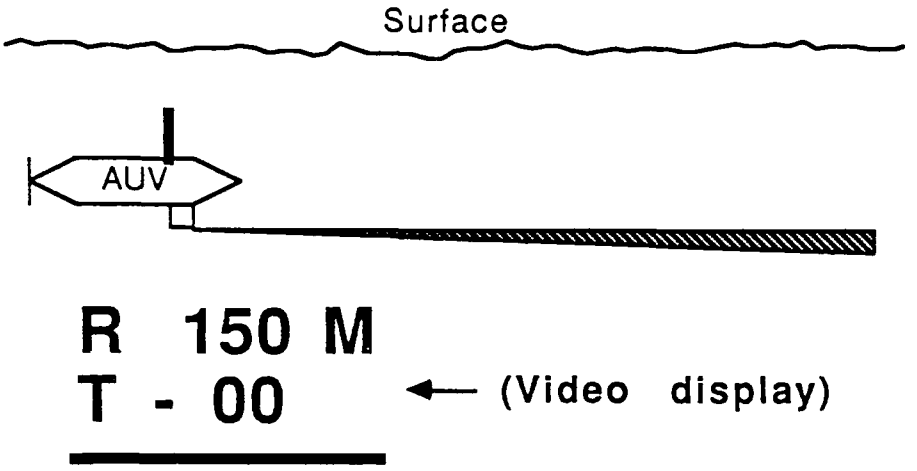
Figure 4.3
Scan Storage Array Display Screen
a) Graphics Screen b) Screen Description

will decrease after each 360° azimuth scan. When all initial AUV and ocean current attributes have been set, press the left mouse button.

The WESMAR sonar video and control panel appear on the graphics screen, along with the ocean floor contour chart and the AUV instrument panel. The video display may be controlled by *turning* the appropriate dials on the sonar control panel, which is accomplished by turning the corresponding dials on the dial box. The AUV course, dive angle, and ocean current direction may also be changed at this point by manipulating the appropriate dial box dial. The AUV speed and ocean current magnitude may also be changed as described earlier.

To *energize* the sonar simulator, turn the power dial to the on position. This will bring up the WESMAR video screen. When the volume dial is turned up, the sonar will *drop* into its operating position below the AUV and begin to scan.

The sonar tilt angle may be manipulated by utilizing the tilt toggle on the sonar control panel. The toggle location, up or down, may be changed by placing the cursor over the tilt toggle and pressing the middle mouse button. This will place the toggle in the opposite position. Once the toggle is in the proper position, place the cursor over tilt toggle and press the right mouse button until desired tilt angle is achieved. Tilt angle information is displayed in the upper left hand corner of the sonar video screen. As Figure 4.4 shows, when the tilt angle is 0°, the sonar beam sweeps parallel to the ocean surface and when the tilt angle is -90°, the sonar beam points straight down. The scan heading may be changed in the identical way. Place the scan direction toggle in the desired position as described above and then press right mouse button until the desired location is reached. The scan line on the sonar screen will change as you hold down the right mouse button. To utilize the range ring indicator, place the range ring direction toggle in the desired position and press the right mouse button until the range ring



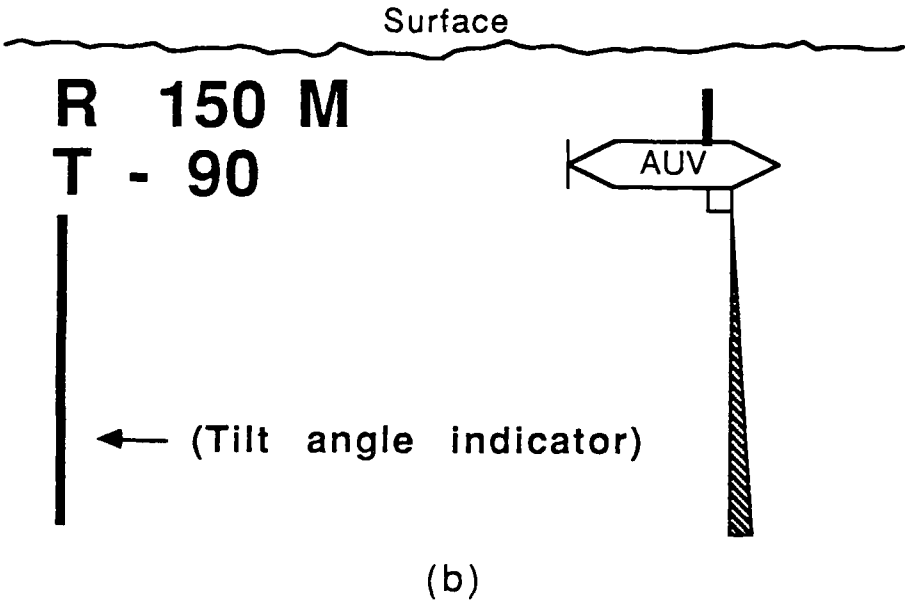


Figure 4.4
Tilt Angle Illustration
a) Tilt Angle = 0° b) Tilt Angle = -90°

reaches the desired location. The horizontal distance and depth below the AUV of any object that the range ring touches may now be read in the upper right hand corner of the video. Figure 4.5 shows an example of these horizontal and depth numbers.

To change the sector scan of the sonar video, turn the sector dial until the desired sector is obtained. The sector determines the area of scan. If the sector is 90°, the sonar will sweep 45° to the left and then 45° to the right of the sonar scan line. The sonar sweep will continually alternate between clockwise and counterclockwise until the sector is reset to 360° (full video). This is a useful capability when the vessel is in a narrow

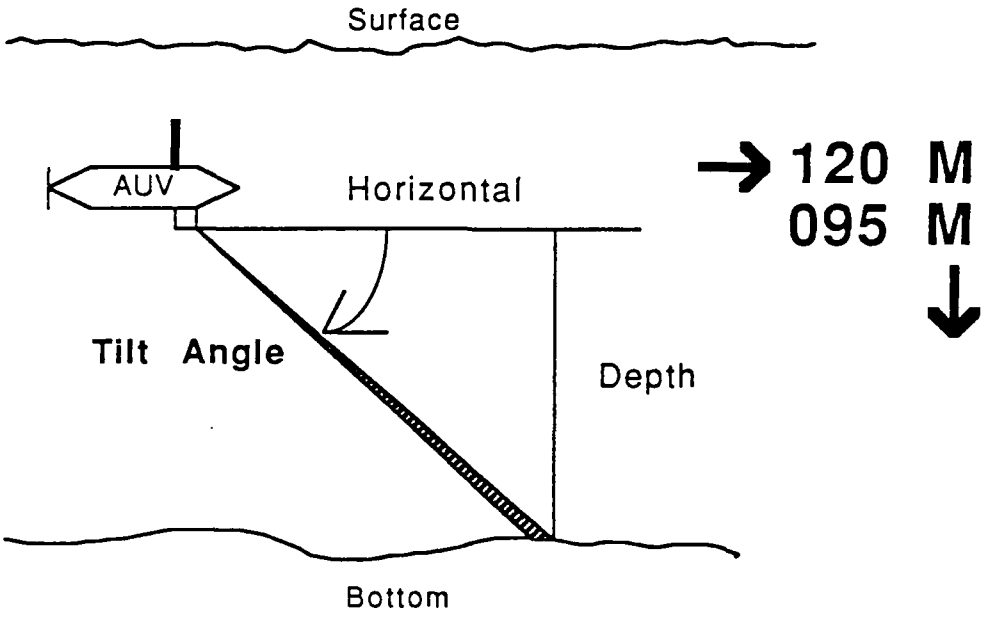


Figure 4.5
Horizontal and Depth Numerical Display

channel. A 15° or 30° sector can be displayed directly in front of the submarine, so the sonar sweeps only the narrow channel ahead. The sonar range may be changed by turning the range selection dial. The minimum range is 15 meters and the maximum range is 800 meters.

The AUV's position or depth may be changed at any time during the scanning process by pressing the left mouse button. The initial settings screen will appear again. If the current scan mode is 3, the successive scan depth arrays will both be reinitialized and all previous scan data will be lost. The AUV's course, speed and dive angle along with the ocean current set and drift may be changed at any time during the scanning process just as described above. During the scanning process, three arrows will appear over the bottom contour map in the upper right hand corner. These arrows represent the AUV's course and speed (green), the ocean current set and drift (yellow) and the AUV's course over ground and speed over ground (red). The magnitude and direction of these arrows will be updated as direction and speed changes are made to the AUV and/or the ocean current.

The sonar scanning process can be temporarily halted, when in modes 2 or 3, in order to bring up the display of successive scans. This is accomplished by pressing the *s* key during scanning. To resume sonar scanning, both after a temporary halt or a complete scan, press the *q* key. Also, if in modes 2 or 3, the AUV's position will not be updated until a scan is completed.

The sonar video return is color coded. Dark blue signifies the water column; no contacts present. Red signifies a contact with the bottom. The light blue color outside of the contact represents sonar shadow. The grey color represents the area outside of the one kilometer square operating area.

Exiting from the program must be executed while the sonar is scanning. To exit, press the *e* key on the graphics station keyboard.

E. SUMMARY

In this chapter, the terrain model, using *Fort Hunter-Liggett* elevation data, was discussed along with the computer simulation program. A comprehensive Computer Simulation User's Manual was also included. In the next chapter, a discussion is presented on how the terrain model and simulation program were used to test the central hypotheses of this thesis.

V. SIMULATION RESULTS

A. INTRODUCTION

In the previous chapters, the mathematical models used to simulate the motion detection problem and the proposed solution were presented in detail. These models were developed in order to determine if an AUV could track its motion using frame to frame correlation of bottom-scanning sonar return data. After development of the computer simulation models, numerous simulation runs were needed to validate these models. Once validated, the individual models were incorporated into one large simulation system needed to test the above hypothesis.

First, preliminary runs of this system were conducted to determine if AUV motion could be detected at all. Once it was determined that AUV motion could be detected, numerous follow up runs were conducted to determine the accuracy of the motion detection mechanism under different environmental and system configurations. This chapter describes these experiments in more detail, presents the results and finally discusses conclusions drawn from these results.

B. DESCRIPTION OF EXPERIMENTS

The simulated motion detection accuracy testing focused on two main experiments. The first was to determine the effect of tilt increment size on accuracy performance (180 runs conducted), while the second was designed to show the effect of AUV displacement on performance (120 runs conducted). Before starting experimentation, a definition of *accuracy* was developed. It was decided that the simulation result of motion detection would be considered *accurate* if the scan comparison was within one scan array cell, either up or down or on either side. If the scan comparison met this criteria, the result

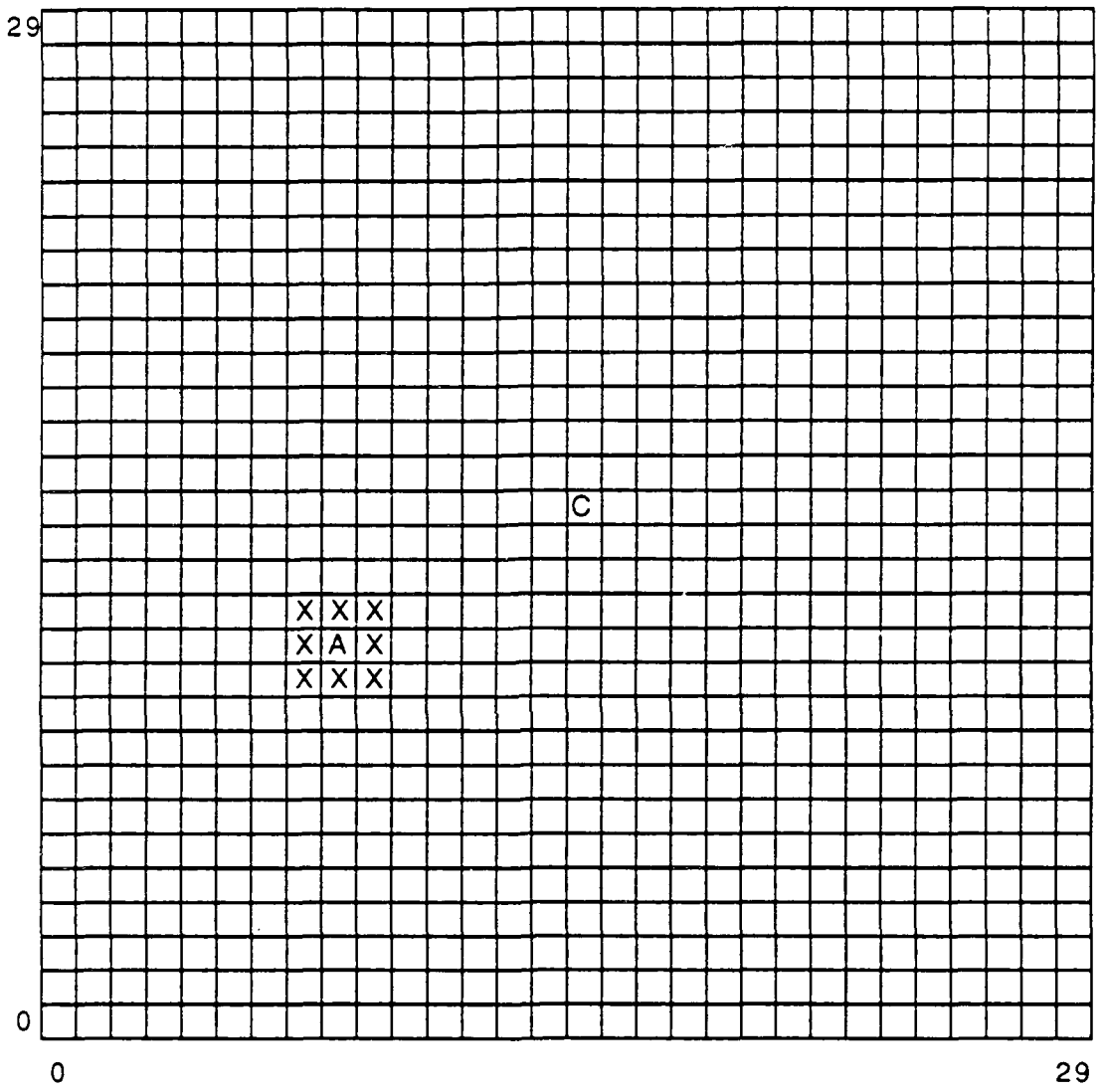
was designated *locked on*. (The comparison of the new and old scans *locked on* to the actual AUV motion.) Figure 5.1 shows this *lock on* situation more clearly. Since the scan comparison shifts the old scan array over the new scan array in array index increments, the actual AUV direction, COG (Course Over Ground), and magnitude, DOG (Distance Over Ground), between scans must be converted to array indices in order for this actual AUV motion to be compared with the final array vertical and horizontal displacement. The A in Figure 5.1 represents the *actual* position of the AUV, at the time the new scan was taken, with respect to the old scan, converted to array indices. To calculate the horizontal and vertical distance of the actual AUV motion in array index units, Equations 5.1 and 5.2 are utilized:

$$horizontal\text{-}distance = round\left(\frac{\sin(actual\text{-}COG) \times actual\text{-}DOG}{beam\text{-}increment}\right) \tag{5.1}$$

$$vertical\text{-}distance = round\left(\frac{\cos(actual\text{-}COG) \times actual\text{-}DOG}{beam\text{-}increment}\right) . \tag{5.2}$$

The beam-increment, in the above Equations, represents the distance of one array index. The C in Figure 5.1 represents the center cell of the new scan. If the old scan center cell's final location is over A or one of the Xs, one cell away from A, the scan comparison *locked on* to the actual AUV motion.

Once the accuracy or *lock on* criteria was determined, the experiments were designed. A quick observation showed that many factors, attributes and parameters were changeable and that any single one may affect the experimentation results. To limit the scope of the experiment, arbitrary or randomized values were assigned to all variables except the one being focused upon. In both experiments, the range setting of the sonar and the depth below the AUV were set at 150 and 50 meters respectively. This allowed for a maximum of 89% of the available scan depth array cells to be used, assuming a fairly flat bottom. Figure 5.2 shows how to calculate this percentage. In general,



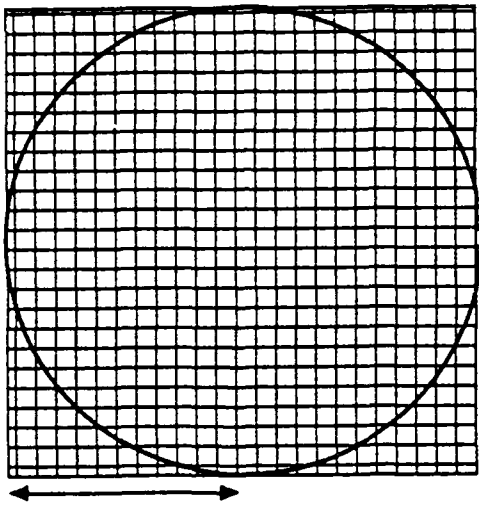
30 x 30 Bottom scan depth storage array

A: Actual direction & distance from center

X: Direction & distance considered "Locked on"

Figure 5.1
"Lock On" Illustration

Scan Depth Array

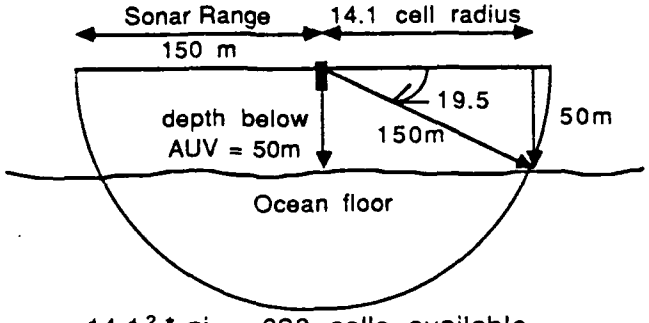


15 cell radius

Cells inside circle = accessible cells = pi * r^2

15^2 * pi ≡ 706 accessible cells

Depth Below AUV/Sonar Range



14.1^2 * pi ≡ 628 cells available

(628 / 706) * 100 = 89 %

89 % of accessible scan depth array cells can be used when sonar range = 150m and depth below AUV = 50m

Figure 5.2
Effect of Depth Below AUV/Sonar Range Ratio on Percentage Availability of Accessible Depth Array Cells

assuming a relatively flat bottom, the smaller the ratio of depth below the AUV to sonar range, the larger the percentage of available scan array cells. This ratio could effect *lock on* accuracy, since the more cells available to be filled in the scan depth array, the more information the scan comparison machinery has to work with. (Two large contour charts are more easily compared and more accurately matched.) Since the number of these possible ratios is infinite, the arbitrary ratio of 50 meters (Depth below AUV)/ 150 meters (Sonar range) was chosen.

With the sonar range setting at 150 meters, the sonar beam increment is 10 meters (1/15 of range setting) and therefore a scan depth array index represents 10 meters. Whenever the top array is shifted one cell, this shift corresponds to 10 meters. When the top array shifts 3 cells right and 2 cells left, this shift corresponds to 30 meters right and 20 meters left. The final array shifts can also be converted in order to compare them with the actual AUV direction and distance traveled between scans by using Equations 5.3 and 5.4.

$$estimated\text{-}COG = arctan\left(\frac{vertical\text{-}cell\text{-}moves}{horizontal\text{-}cell\text{-}moves}\right) \tag{5.3}$$

$$estimated\text{-}DOG = beam\text{-}increment\ \sqrt{vertical\text{-}cell\text{-}moves^2 + horizontal\text{-}cell\text{-}moves^2} \tag{5.4}$$

This ability to compare estimated COG and DOG with actual COG and DOG is not used to determine accuracy in the experiments conducted in this thesis. Rather, the above definition of *lock on* is used.

The AUV's heading was held constant at due north. This allowed the scan depth array display, which is relative to the AUV, to have north always toward the top of the screen. Since the scan array display and the bottom contour chart were both displayed with north toward the top of the screen, the two images could be compared without rotating one to match the other. The AUV dive angle was set at zero so motion would

only occur in the x - y plane. The AUV's speed was also set at zero for all experimentation, so the motion detected would be only that motion generated by the ocean current.

A major assumption in the two experiments was that AUV motion was negligible during the complete bottom scan cycle. The AUV remained stationary during the first complete scan of the bottom, was then moved the desired direction and distance, and then remained stationary during the second complete bottom scan.

C. EFFECT OF TILT INCREMENT ON PERFORMANCE

As stated in Chapter III, the smaller the tilt increment, the more bottom scan depth array cells are filled with depth information. It seems reasonable that the greater thenumber of cells in consecutive scans, the better the chances are for *lock on*. To test this conjecture, with the sonar range, depth below AUV, AUV course, speed and dive angle set to the constant values described earlier, 10 starting locations were chosen at random. At each random location, a corresponding, pseudo random (36° increments), ocean current direction was assigned. (Location one was assigned a current direction of 0°, two was assigned 36°, etc.) The first segment of the experiment assigned one knot to the ocean current magnitude, which corresponded to an AUV displacement of approximately ten meters. Then at each starting location, with its corresponding set direction, lock on results were observed when the tilt increment value was set at one degree. Results were also observed for tilt increments of 2°, 4°, 8°, 16° and finally 32°.

The second segment of the experiment assigned two knots to the ocean current magnitude, which corresponds to an AUV displacement between scans of 20 meters. Lock on results were then obtained at each location and tilt increment amount described in the experiment's first segment. The third segment observed lock on results using the

same locations and tilt increments, but by generating an AUV displacement of 40 meters between scans.

The results of this experiment are displayed in Figure 5.3 and were as expected. Generally, the smaller the tilt increment, the more information per scan, and the higher

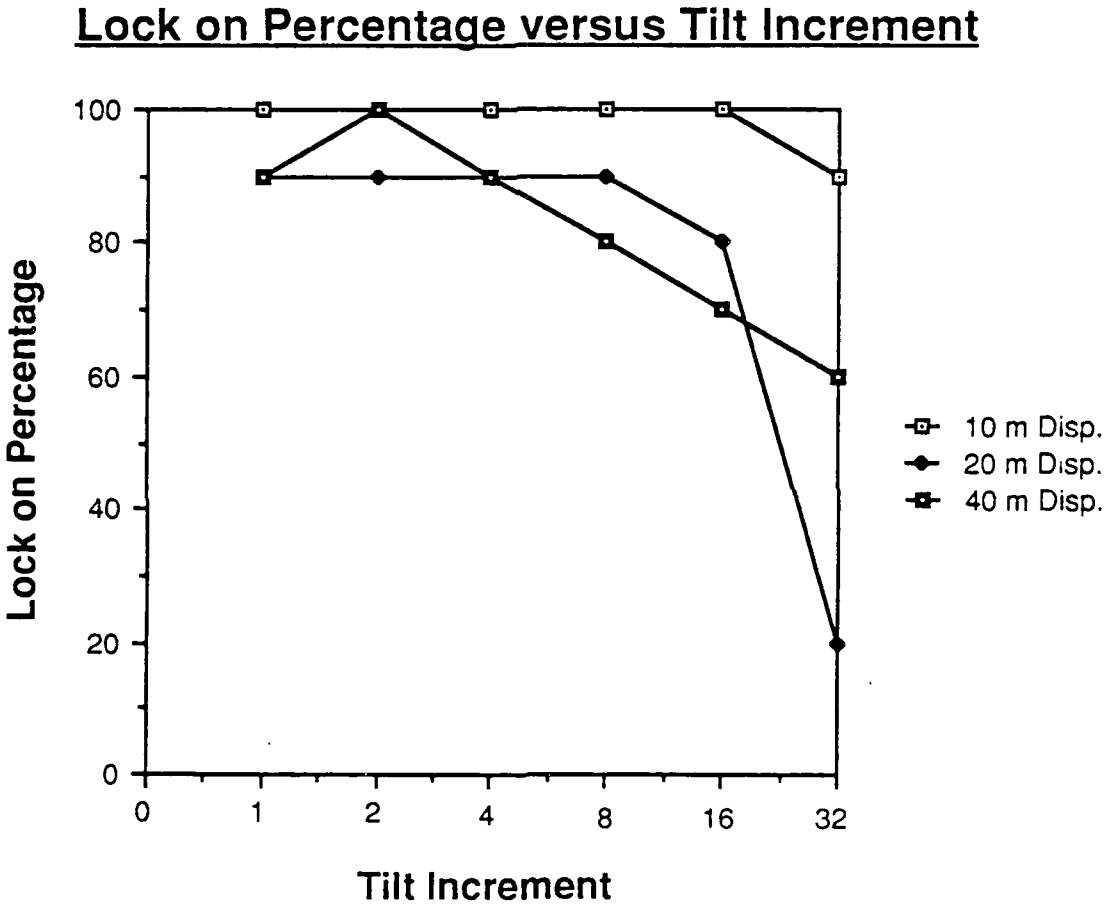


Figure 5.3
Percentage of Lock-on versus Tilt Increment

the percentage of lock on. Another interesting results was that the smaller the displacement between scans, the higher the lock on percentage.

D. EFFECT OF DISPLACEMENT ON PERFORMANCE

This second experiment measures motion detection accuracy while changing AUV displacement over different types of terrain. Since the scan depth array comparison relies on the difference between depth values to determine the best match between the two scans, the larger the AUV displacement between scans over a completely flat terrain, the smaller the lock on percentage. Consecutive scans over flat terrain will match perfectly at any overlaying configuration. Thus it is to be expected that the flatter the terrain, the less accurate the scan comparison would be.

To test this theory, the sonar range setting, depth below AUV, AUV heading, speed and dive angle were set to the values noted earlier. An arbitrary tilt increment value of four degrees was chosen, because this tilt increment proved fairly accurate in the previous experiment. Four AUV displacement values were then chosen: 5, 10, 20 and 40 meters, along with three different starting locations of different terrain types. Terrain one was a ridge, terrain two was irregular, and terrain three was relatively flat.

The first segment of the experiment used the ridge starting location and observed results at each displacement and set direction (0° to 360° in 36° increments). The lock on percentage at 5, 10 and 40 meter displacements was 100 while at 20 meters, it was 90. When the second (irregular terrain) and third (relatively flat) segments of the experiment were conducted, the lock on percentages for all four displacements were 100.

These results were certainly not expected. It was not the case that the flatter the terrain, the lower the lock on percentage. It was also not the case that the larger the AUV displacement, the lower the lock on percentage. The lock on percentage proved very

high for any of the three terrains with any of the four AUV displacements. The reasons for these apparently anomolous results are unknown at present.

E. SUMMARY

The results in this chapter are extremely favorable toward the hypothesis that an AUV can detect its own motion by using only bottom tracking sonar. The results also show that the mathematical models developed in the earlier chapters are useful for modeling synthetic sonar images and using frame to frame correlation of these successive images. However, much more work is needed to obtain statistical information about how scan comparison accuracy varies with different environmental conditions and AUV configurations before any degree of confidence can be attached to the hypothesis of this work. Regardless of the results of such a study, however, this research clearly provides a viable basis for AUV motion detection.

VI. SUMMARY AND CONCLUSIONS

A. RESEARCH CONTRIBUTIONS

As described in the abstract, an AUV must have the capability to detect vehicle motion in order to keep station. Unfortunately, the only self contained, accurate motion determination devices aboard submarines are Inertial Navigation Systems. Motion may also be detected by trained personnel monitoring a sonar or fathometer, but these sensors in their present configuration, are not nearly accurate enough for station keeping. Since Inertial Navigation System accuracy is generally proportional to the size of the system (not to mention the cost), an alternative method for accurate motion determination by a relatively small, inexpensive system is needed.

This research is the first step in showing that AUV motion can be accurately detected using only bottom tracking sonar. The WESMAR sonar is relatively inexpensive, small and could be used in an experimental AUV. This research has also contributed information needed in determining the minimal AUV sensor suite.

Another important product of this work is the development of the WESMAR sonar graphics simulator. This model can be used as a working tool or test-bed to incorporate related research work in the future.

B. RESEARCH EXTENSIONS

The size and complexity of the WESMAR sonar simulator required that simplified assumptions and conditions be made in order to complete this first portion of research during the allotted time. Follow on work could be conducted to bring the simulation closer to "real life." Noise could be added to the sonar beam propagation through the water or AUV movement could be added during the scanning process. Other extensions

could be studying different bottom-tracking sonars to determine if different systems perform better than others, incorporating doppler sonar in the simulation model to enhance motion sensing accuracy, testing the sonar for total bottom navigation, utilizing the bottom-tracking sonar for INS correction and using new sonar technology, such as phased array sonar, to increase speed and accuracy of motion detection. One extension could be to test the motion detection model developed in this thesis with actual sonar return data. Another extension could be to investigate the assumption that when an accurate ocean floor chart is available, accurate motion control as well as station keeping could be possible.

One interesting extension could be to develop a new computer simulation that would replace the mechanical sonar scanner with a 2D array of hydrophones. The simulation would be used to determine if 2D correlation (grid search) on single pulse returns could be used to correct INS drift. This CVL broad beam and hydrophone array approach could be used to determine AUV position and not just velocity.

A final extension could be to couple this motion detection model with an high level AUV *intelligent* control model, where the output of the motion detection model would be used by the control model to perform AUV speed and direction changes to constantly counteract the ocean current forces. This could have a significant impact on the development of a viable AUV in the future.

LIST OF REFERENCES

1. Bane, G. L. and Ferguson, J., "The Evolutionary Development of the Military Autonomous Underwater Vehicle," *Proceedings of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, v. 1, pp. 60-88, June 1987.
2. Klevebrant, H. and Lindström, B., "A User's Requirements on an Autonomous Underwater Vehicle System and a Proposed Development System," *Proceedings of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, v. 1, pp. 44-59, June 1987.
3. McFarlane, J. R., Jackson, E., and Hartley, P., "Robotic Marine Vehicles for Surveying," *Unmanned Systems*, v. 5, no. 4, pp. 17-23, Spring 1987.
4. Butler, B. and Maryka, S., "Evolution of the Dolphin Multi-Vehicle Control System," *Proceedings of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, v. 1, pp. 23-32, June 1987.
5. Freund, J. F., *New Technologies in U.S. Navy Undersea Vehicles,* National Ocean Systems Center, San Diego, CA, February 1987.
6. Michel, T. L. and Conway, T., "EPAULARD: Operational Developments," *Proceedings of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, v. 1, pp. 14-17, June 1987.
7. Jalbert, J., "Low Level Architecture for the New EAVE Vehicle," *Proceedings of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, v. 1, pp. 238-244, June 1987.
8. Hackman, W. D., *Seek & Strike: Sonar, Anti-submarine Warfare and the Royal Navy 1914-1954,* Her Majesty's Stationery Office, London, England, 1984.
9. Williams, W. P., "Sonar Navigation Aids," in *Navigation Systems: A Survey of Modern Electronic Aids*, ed. G. E. Beck, pp. 284-310, Van Nostrand Reinhold Co., London, England, 1971.
10. General Electric Company: Undersea Systems Department, *Correlation Velocity Log Features*, Syracuse, NY.
11. Bookheimer, W. C. and Edward, J. A., *Navigation Sonar for Deep Water Operation: GE Correlation Velocity Log Model QV-12,* General Electric Company: Undersea Systems Department, Syracuse, NY.
12. Dickey, F. R. Jr., Bookheimer, W. C., and Rhoades, K. W., "Implementation and Testing of a Deep Water Correlation Velocity Sonar," *Proceedings of the Fifteenth Annual Offshore Technology Conference*, pp. 437-443, May 1983.
13. Dickey, F. R. Jr. and Edward, J. A., "Velocity Measurement Using Correlation Sonar," *Proceedings of the Position Location and Navigation Symposium*, November 1978.

14. Johnson, D. and Eppig, S., "Aided Inertial Navigation Systems for Underwater Vehicles," *Proceedings of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, v. 1, pp. 265-282, June 1987.

15. Klass, P. J., "Inertial Navigation: Out of the Laboratory into Missle Systems," *Aviation Week Exclusive: Inertial Guidance*, pp. 4-6, 1956.

16. Putman, R. G., *A Conceptual Design of an Inertial Navigation System for an Autonomous Submersible Testbed Vehicle,* M.S. Thesis, Naval Postgraduate School, Monterey, CA, September 1987.

17. Draper, C. S., Wrigley, W., and Hovorka, J., "Inertial Guidance," in *International Series on Aeronautical Sciences and Space Flight*, pp. 1-23, Pergamon Press, 1960.

18. Parvin, R. H., "Inertial Navigation," in *Principles of Guided Missle Design*, ed. G. Merrill, pp. 4-13, D. Van Nostrand Company, Inc., 1962.

19. Geyer, E. M. Jr. and D'Appolito, J., "Characteristics and Capabilities of Navigation Systems for Unmanned Untethered Submersibles," *Proceedings of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, v. 1, pp. 320-347, June 1987.

20. Anon., *SS256 Digital Color Sonar Owner's Manual,* WESMAR Marine Electronics, Bothell, WA.

21. Rickenbach, M. D., *Correction of Inertial Navigation System Drift Errors for an Autonomous Land Vehicle Using Optical Radar Terrain Data,* M.S. Thesis, Naval Postgraduate School, Monterey, CA, September 1987.

22. Anon., *WESMAR Digital Color Sonar Information Pamphlet,* WESMAR Marine Electronics, Bothell, WA.

23. Fu, K. S., Gonzales, R. C., and Lee, C. S., *Robotics: Control, Sensing, Vision and Intelligence,* McGraw Hill, 1987.

24. McGhee, R. B. and Ross, R. S., *Robotics: A Kinematics Perspective,* Lecture notes for CS4313, Naval Postgraduate School, Monterey, CA, October 1987.

25. Poulus, D., *Range Image Processing for Local Navigation of an Autonomous Land Vehicle,* M.S. Thesis, Naval Postgraduate School, Monterey, CA, September 1986.

26. Anon., *IRIS User's Guide,* Silicon Graphics, Inc., Mountain View, CA, 1987.

27. Anon., *Unix Programmer's Manual Volume IA: Commands,* Silicon Graphics, Inc., Mountain View, CA, 1987.

28. Anon., *Unix Programmer's Manual Volume IB: Administration, Miscellaneous, and Options,* Silicon Graphics, Inc., Mountain View, CA, 1987.

29. Anon., *Unix Programmer's Manual Volume IIB: Languages and Tools,* Silicon Graphics, Inc., Mountain View, CA, 1986.

APPENDIX - PROGRAM LISTING

```
/*+-------------------------------+
 |
 |      main_sonar.c
 |
 +-------------------------------+              */

/* main_sonar -- an IRIS-2400 program by Chet Hartley. This program reads in a
  1km x 1km section of a terrain data (Fort Hunter Ligett) and then builds a sonar
  to simulate WESMAR SS265 sonar output.
   It then can make successive scans of the ocean floor, analyze them and then determine
  in which direction and how far the AUV moved. October 1987-June 1988 */

#include "gl.h"          /* get the graphics defs          */
#include "device.h"      /* get the graphics device defs   */
#include "sonar.h"       /* constants                      */
#include "stdio.h"

/* FHL elevations used as depths     */
int bottom_data[BOTTOM_POINTS_WIDTH][BOTTOM_POINTS_HEIGHT];
/* arrays to hold successive sonar scan data */
int right_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
int left_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];

/* max and min depth vars for chart generation */
int max_depth = 0;
int min_depth = 9999;

main()
{
 /* initial submarine location and orientation values */
 float x_coord = 469.0; /* meters */
 /* the y_coord is negative due to orthoganal coord system choice */
 float y_coord = -267.0; /* meters */
 /* depth is passed in meters */
 float auv_depth = 462.0;
 /*float auv_depth = M_PER_FEET * 500.0;  feet converted to meters */
 int course = 0;
 float speed = 0.0;
 int dive_angle = 0;
 int roll_angle = 0;

 /* variables for changing course, speed and dive_angle */
 int selected_course = 0;
 float selected_speed = 0.0;
 int selected_dive_angle = 0;

 /* initial current values */
 int set = 0;
 float drift = 1.0;

 /* initial cog/sog values */
 /* cog = course over ground, sog = speed over ground */
```

```
int cog = 0;
float sog = 0.0;

/* dial locations read from the dial box */
int power_dial_location=0;
int volume_dial_location=0;
int sector_dial_location=0;
int range_dial_location=0;

/* booleans for dial changes */
short power_dial_change=FALSE;
short volume_dial_change=FALSE;
short sector_dial_change=FALSE;
short range_dial_change=FALSE;
short change_location = FALSE;


/* boolean for hoist , down is operational */
short hoist_down=FALSE;

/* boolean for power on */
short power_on=FALSE;

/* initial wesmar sonar configuration */
int sector_setting = 360;
short sector_setting_change=FALSE; /* boolean for sector change */
int previous_sector_setting= 360;
int range_setting = 150;
short range_setting_change=TRUE; /* bool for range setting change */
int previous_range_setting= 800;

short tilt_toggle_up=FALSE; /* bools for toggles */
short scan_toggle_up=TRUE;
short range_toggle_up=FALSE;

int tilt_angle = 0;
int tilt_inc = 1;
short tilt_angle_change = FALSE;
int scan_heading = 0;
short scan_heading_change = FALSE;
int range_ring_location = 0;

short sweep_clockwise = TRUE; /* bool for sweep direction */
int sonar_sweep_location = 0;
int sector_sweep_start = 0;
int sector_sweep_end = 0;

/* the sonar beam length */
float beam_length = 0.0;
float beam_inc = BEAM_INC_800;
short encountered_contact = FALSE;
short scan_complete = FALSE;
short scan_mode = COMPLETE_SCAN_AND_STORE;
```

```
/* boolean for 3 buttons hit on mouse */
short exit = FALSE;

/* boolean for AUV running aground */
short aground_flag = FALSE;

/* boolean for active array toggle */
/* start with left array active */
short left_depth_array_active = TRUE;

/* vars for holding depth array max/mins */
int right_array_max_depth, right_array_min_depth;
int left_array_max_depth, left_array_min_depth;

/* vars for holding depth array AUV coords */
float left_x_coord, left_y_coord, left_depth;
float right_x_coord, right_y_coord, right_depth;

/* boolean for displaying depth arrays before sweep has completed */
short request_depth_array_display = FALSE;

/* make sonar_video,sonar_ctrl_panel  */
Object sonar_sweep, /* objects for parts of wesmar */
 scan_heading_line,
 scan_mask,
 dials,
 tilt_angle_indicator,
 video_arc_array[45], /* 45 8 degree "filled arcs" on video screen */
 chart;  /* chart showing position of auv */

Tag power_tag,  /* tags for dial rotations */
 volume_tag,
 sector_tag,
 range_tag,
 scan_mask_tag, /* tags for other parts rotations */
 tilt_angle_tag,
 scan_heading_tag,
 sonar_sweep_tag,
 arc_tag_array[45][15]; /*tags to change colors out on 8 degree arcs*/

FILE *fopen(), *fp;

Colorindex wmask;

/* initialize the iris */
init_iris();

/* enable bitplanes */
wmask = (1<<getplanes()) -1;
writemask(wmask);

/* read the data file into the bottom_data_array */
load_bottom_data();
```

```
/* make wesmar objects */
make_video_rtn(video_arc_array, arc_tag_array);
make_dials(&dials, &power_tag, &volume_tag, &sector_tag, &range_tag);
make_scan_mask(&scan_mask, &scan_mask_tag);
make_tilt_angle_indicator(&tilt_angle_indicator,&tilt_angle_tag);
make_scan_heading_line(&scan_heading_line,&scan_heading_tag);
make_sonar_sweep(&sonar_sweep,&sonar_sweep_tag);
make_chart(&chart);

/* get initial settings */
get_auv_settings(&scan_mode, &x_coord, &y_coord, &auv_depth, &course, &speed,
 &dive_angle, &selected_course, &selected_speed, &selected_dive_angle, &set, &drift,
 &cog, &sog, chart, &aground_flag, &tilt_inc);

/* initialize the depth arrays */
initialize_depth_arrays(&sonar_sweep_location, &left_depth_array_active,
 &left_array_max_depth, &left_array_min_depth, &right_array_max_depth,
 &right_array_min_depth, x_coord, y_coord, auv_depth,
 &left_x_coord, &left_y_coord, &left_depth, &right_x_coord, &right_y_coord, &right_depth);

color(BLACK);
clear();

/* ready chart for upper right hand corner */
editobj(chart);
 objreplace(STARTTAG);
  viewport(647,1023,391,767);
  linewidth(5);
closeobj();

/* put chart up in upper right hand corner */
callobj(chart);

/*put up sonar in upper left corner */
make_sonar_video(sonar_sweep, range_setting, tilt_angle, range_ring_location, hoist_down,
 power_on, x_coord, y_coord, auv_depth, tilt_angle_indicator, scan_mask, scan_heading_line,
 range_setting_change, sector_setting, video_arc_array, arc_tag_array, course, dive_angle,
 roll_angle, &sonar_sweep_location, sweep_clockwise, &beam_length, &beam_inc,
 &encountered_contact);

/* put up the auv instrument readings */
make_readout(scan_mode, x_coord, y_coord, auv_depth, course, speed,
 dive_angle, set, drift, cog, sog, tilt_inc);

/* put up the sonar control panel along the bottom of the screen */
make_sonar_panel(hoist_down, tilt_toggle_up, scan_toggle_up, range_toggle_up,
 volume_dial_location, power_dial_location, sector_dial_location, range_dial_location, dials);

swapbuffers();

/* put the chart in the upper right hand corner */
callobj(chart);

/* ready chart for upper left hand corner */
```

```
editobj(chart);
 objreplace(STARTTAG);
  viewport(0,647,120,767);
  linewidth(9);
closeobj();

while(TRUE)
{
 /* read the mouse, keyboard and dials for input */
 read_controls(&scan_mode, &power_dial_change, &volume_dial_change, &sector_dial_change,
  &range_dial_change, &sector_setting_change, &tilt_angle_change, &scan_heading_change,
  &previous_sector_setting, &previous_range_setting, &power_on, &hoist_down,
  &range_setting_change, &power_dial_location, &volume_dial_location,
  &sector_dial_location, &sector_setting, &range_dial_location, &range_setting,
  &tilt_angle, &tilt_inc, &scan_heading, &range_ring_location, &scan_toggle_up,
  &tilt_toggle_up, &range_toggle_up, &exit, &selected_course, &selected_speed,
  &selected_dive_angle, &set, &drift, &change_location, &request_depth_array_display);

 if (exit)
  break;
 else
 {
  if (change_location || aground_flag)
  /* get the new location and auv settings */
  {
   get_auv_settings(&scan_mode, &x_coord, &y_coord, &auv_depth, &course, &speed,
    &dive_angle, &selected_course, &selected_speed, &selected_dive_angle, &set, &drift,
    &cog, &sog, chart, &aground_flag, &tilt_inc);

   /* initialize the depth arrays */
   initialize_depth_arrays(&sonar_sweep_location, &left_depth_array_active,
    &left_array_max_depth, &left_array_min_depth, &right_array_max_depth,
    &right_array_min_depth, x_coord, y_coord, auv_depth, &left_x_coord, &left_y_coord,
    &left_depth, &right_x_coord, &right_y_coord, &right_depth);

   /* ready chart for upper right hand corner */
   editobj(chart);
    objreplace(STARTTAG);
    viewport(647,1023,391,767);
    linewidth(5);
   closeobj();

   /* put chart in upper right hand corner */
   callobj(chart);

   swapbuffers();

   /* put chart in upper right hand corner */
   callobj(chart);

   /* ready chart for upper left hand corner */
   editobj(chart);
    objreplace(STARTTAG);
     viewport(0,647,120,767);
```

```
  linewidth(9);
 closeobj();
}


/* edit the control panel dial marks */
edit_dials(dials, power_dial_location,volume_dial_location, sector_dial_location,
 range_dial_location, power_dial_change, volume_dial_change, sector_dial_change,
 range_dial_change, volume_tag, power_tag, sector_tag, range_tag);

if (scan_heading_change)
 /* edit the scan heading line */
 edit_scan_heading_line(scan_heading_line, scan_heading, scan_heading_tag);

/* edit the sonar scan mask */
edit_scan_mask(scan_mask, scan_heading, scan_heading_change,
 sector_setting_change, sector_setting, scan_mask_tag);

/* edit the tilt angle indicator on the video display */
edit_tilt_angle_indicator(tilt_angle_indicator, &tilt_angle_change, tilt_angle, tilt_angle_tag);

/* put up the sonar video screen */
make_sonar_video(sonar_sweep, range_setting, tilt_angle, range_ring_location, hoist_down,
 power_on, x_coord, y_coord, auv_depth, tilt_angle_indicator, scan_mask, scan_heading_line,
 range_setting_change, sector_setting, video_arc_array, arc_tag_array, course, dive_angle,
 roll_angle, &sonar_sweep_location, sweep_clockwise, &beam_length, &beam_inc,
 &encountered_contact);

/* store the sonar return data in the depth arrays */
store_return_data(scan_mode, encountered_contact, auv_depth, sonar_sweep_location,
 beam_length, beam_inc, tilt_angle, left_depth_array_active, &left_array_max_depth,
 &left_array_min_depth, &right_array_max_depth, &right_array_min_depth);

/* put up the auv instrument readings */
make_readout(scan_mode, x_coord, y_coord, auv_depth, course, speed, dive_angle,
 selected_course, selected_speed, selected_dive_angle, set, drift, cog, sog,
 tilt_inc);

/* put up the sonar control panel */
make_sonar_panel(hoist_down, tilt_toggle_up, scan_toggle_up, range_toggle_up,
 volume_dial_location, power_dial_location, sector_dial_location, range_dial_location,
 dials);

writemask(CHART_MASK);
/* make the direction and velocity arrows using
 the CHART_MASK so we don't have to rebuild the chart
 each time */
make_arrows(x_coord, y_coord, auv_depth, course, speed, set, drift, cog, sog,
 sonar_sweep_location, beam_length, tilt_angle, range_setting);

/* change writemask back */
writemask(wmask);

swapbuffers();
```

```
/* edit the sonar sweep on the video screen */
edit_sonar_sweep(sonar_sweep, &sweep_clockwise, &sonar_sweep_location,
 &sector_sweep_start, &sector_sweep_end, sector_setting_change, sector_setting,
 scan_heading, scan_heading_change, sonar_sweep_tag, power_on, hoist_down);

/* check scan mode and determine if a scan has completed */
bottom_scan_controller(scan_mode, power_on, hoist_down, &tilt_angle, &tilt_angle_change,
 tilt_inc, sonar_sweep_location, &scan_complete);

/* display the depth arrays if sweep completed */
display_depth_arrays(scan_mode, drift, tilt_inc, tilt_angle, scan_complete,
 request_depth_array_display, range_setting, beam_inc, left_depth_array_active,
 left_array_max_depth, left_array_min_depth, right_array_max_depth, right_array_min_depth,
 left_x_coord, left_y_coord, left_depth, right_x_coord, right_y_coord, right_depth);

/* calculate the next position of the auv */
get_next_position(scan_mode, &x_coord, &y_coord, &auv_depth, &cog, &sog, &course,
 &speed, &dive_angle, selected_course, selected_speed, selected_dive_angle, set,
 drift, &aground_flag, scan_complete);

/* check array toggle for update */
update_depth_arrays(scan_complete, x_coord, y_coord, auv_depth, &left_depth_array_active,
 &left_array_max_depth, &left_array_min_depth, &right_array_max_depth,
 &right_array_min_depth, &left_x_coord, &left_y_coord, &left_depth, &right_x_coord,
 &right_y_coord, &right_depth);

if (scan_complete || request_depth_array_display)
{
 /* ready chart for upper right hand corner */
 editobj(chart);
  objreplace(STARTTAG);
  viewport(647,1023,391,767);
  linewidth(5);
 closeobj();

 /* put chart in upper right hand corner */
 callobj(chart);

 swapbuffers();

 /* put chart in upper right hand corner */
 callobj(chart);

 /* ready chart for upper left hand corner */
 editobj(chart);
  objreplace(STARTTAG);
   viewport(0,647,120,767);
   linewidth(9);
 closeobj();

 /* edit the tilt angle indicator on the video display */
 edit_tilt_angle_indicator(tilt_angle_indicator, &tilt_angle_change, tilt_angle, tilt_angle_tag);
```

```
make_sonar_video(sonar_sweep, range_setting, tilt_angle, range_ring_location, hoist_down,
 power_on, x_coord, y_coord, auv_depth, tilt_angle_indicator, scan_mask,
 scan_heading_line, range_setting_change, sector_setting, video_arc_array, arc_tag_array,
 course, dive_angle, roll_angle, &sonar_sweep_location, sweep_clockwise, &beam_length,
 &beam_inc, &encountered_contact);

/* put up the auv instrument readings */
make_readout(scan_mode, x_coord, y_coord, auv_depth, course, speed, dive_angle,
 selected_course, selected_speed, selected_dive_angle, set, drift, cog, sog, tilt_inc);

/* put up the sonar control panel */
make_sonar_panel(hoist_down, tilt_toggle_up, scan_toggle_up, range_toggle_up,
 volume_dial_location, power_dial_location, sector_dial_location, range_dial_location,
 dials);

writemask(CHART_MASK);
/* make the direction and velocity arrows using
 the CHART_MASK so we don't have to rebuild the chart
 each time */
make_arrows(x_coord, y_coord, auv_depth, course, speed, set, drift, cog, sog,
 sonar_sweep_location, beam_length, tilt_angle, range_setting);

/* change writemask back */
writemask(wmask);

swapbuffers();

} /* end if display arrays */

/* reset all bools to false */
reset_flags(&scan_complete, &power_dial_change, &volume_dial_change,
 &sector_dial_change, &range_dial_change, &scan_heading_change, &sector_setting_change,
 &range_setting_change, &change_location, &encountered_contact,
 &request_depth_array_display);

} /* else not exit */

} /* while */

/* clean up */
color(BLACK);
clear();
swapbuffers();
color(BLACK);
clear();
swapbuffers();
finish();
greset();
gexit();

}  /* end of main_sonar */
```

```
/*+-----------------------------+
 |
 |      sonar.h
 |
 +-----------------------------+

/* This is the file that contains all the constants
   being used in the WESMAR sonar simulator */

#define M_PER_FEET 0.3048 /* meters per foot */
#define FEET_PER_M 3.2808 /* feet per meter */

#define FAR_CENTER_X 62.0 /* far gain dial center */
#define FAR_CENTER_Y 6.0
#define NEAR_CENTER_X 52.0 /* near gain dial center */
#define NEAR_CENTER_Y 6.0

#define DIAL_RADIUS 2.0 /* radius of black dials on panel */

#define START_TOGGLE_Y 6.0 /* y-coord that all toggles start on */

#define TILT 1 /* tilt toggle */
#define SCAN 2 /* scan toggle */
#define RANGE_RING 3 /* range toggle */

#define UP 1 /* for toggle up/down */
#define DOWN 0

#define SWEEP_STEP 8 /* 8 degrees (width of sonar beam)*/

#define POWER_OFF 3 /* if 3 or less, consider power off */

#define VOLUME_OFF 3 /* if 3 or less, consider volume off */

#define SONAR_RADIUS 44.0  /* world coord sonar radius */

#define SONAR_RADIUS_MINUS_A_LITTLE 43.80 /* radius for outer
     ring of range ring overlay */
/* next to outer ring */
#define THREE_QTR_SONAR_RADIUS (SONAR_RADIUS*(3.0/4.0))
/* middle ring */
#define HALF_SONAR_RADIUS (SONAR_RADIUS/2.0)
/* inside ring */
#define ONE_QTR_SONAR_RADIUS (SONAR_RADIUS*(1.0/4.0))

#define RESOLUTION_IN_METERS 12.5  /* depth value every 12.5 m */

/* bottom data max column */
#define BOTTOM_POINTS_WIDTH 80
/* bottom data max row */
#define BOTTOM_POINTS_HEIGHT 80

#define RTOD 57.29578  /* radians to degrees conversion factor   */
#define DTOR 0.0174533 /* degrees to radians conversion factor   */
```

```
/* define color numbers */
/* for wmask */
#define ORANGE 9
#define GREY 10

/* for CHART_MASK */
#define A_MAGENTA 1024
#define A_YELLOW 512
#define A_GREEN 256
#define A_RED 128
#define A_BLACK 64

/* cursor constants */
#define CURSOR_COLOR RED
#define MY_CURSOR 1

/* x and y values for all video read out number locations */
#define RANGE_SETTING_ONES_X -27.0
#define RANGE_SETTING_TENS_X -32.0
#define RANGE_SETTING_HUN_X -37.0
#define RANGE_SETTING_Y 44.0

#define RANGE_RING_ONES_X 39.0
#define RANGE_RING_TENS_X 34.0
#define RANGE_RING_HUN_X 29.0
#define RANGE_RING_Y 44.0

#define TILT_ONES_X -33.0
#define TILT_TENS_X -38.0
#define TILT_Y 38.0

#define DEPTH_ONES_X 39.0
#define DEPTH_TENS_X 34.0
#define DEPTH_HUN_X 29.0
#define DEPTH_Y 38.0

/* number of arcs in one beam */
#define NUMBER_OF_ARCS 15
#define NUMBER_OF_ARCS_MINUS1 (NUMBER_OF_ARCS-1)

/* sonar video colors */
#define NO_CONTACT_COLOR BLUE
#define CONTACT_COLOR RED
#define SHADOW_COLOR CYAN
#define BOUNDARY_COLOR GREY

/* amount the beam tip will be increased/decreased each time
    depending on the range setting */
#define BEAM_INC_800 (800/NUMBER_OF_ARCS) /* 60.0 */
#define BEAM_INC_600 (600/NUMBER_OF_ARCS) /* 30.0 */
#define BEAM_INC_400 (400/NUMBER_OF_ARCS) /* 26.6 */
#define BEAM_INC_300 (300/NUMBER_OF_ARCS) /* 20.0 */
#define BEAM_INC_200 (200/NUMBER_OF_ARCS) /* 13.3 */
```

```
#define BEAM_INC_150 (150/NUMBER_OF_ARCS) /* 10.0 */
#define BEAM_INC_100 (100/NUMBER_OF_ARCS) /* 6.6 */
#define BEAM_INC_60 (60/NUMBER_OF_ARCS) /* 4.0 */
#define BEAM_INC_30 (30/NUMBER_OF_ARCS) /* 2.0 */
#define BEAM_INC_15 (15/NUMBER_OF_ARCS) /* 1.0 */

#define PI 3.1416
#define HALF_PI 1.5708

/* returned when beam tip reaches database boundry */
#define BOUNDARY_FLAG_VALUE 999999

/* max and min auv speeds */
#define MAX_SPEED 12  /* 12 kts */
#define MIN_SPEED -4 /* 4 kts astern */

/* world to screen speed increments on speed bar */
/* SPEED_INC (speed_ortho_bar_range(71-53)/(MAX_SPEED_AHEAD -
 MIN_SPEED)) */
#define SPEED_INC (18/(MAX_SPEED - MIN_SPEED))

/* location in y of the zero mark on the speed bar */
#define ZERO_Y (71.0 - (SPEED_INC * MAX_SPEED))

/* current bounds */
#define MAX_DRIFT 6 /* 6 knots */
#define DRIFT_INC (18/MAX_DRIFT) /* 3 */

/* constants for depth to color conversion */
#define MAX_COLOR_NUMBER 63
#define MAX_DEPTH 2200
#define NUMBER_OF_COLORS 47

/* constants for creating the direction/velocity arrows */
#define ARROW_FACTOR 20
#define ARROW_WING_FACTOR (ARROW_FACTOR * 0.20)

/* constant determining size of the position plus on chart */
#define PLUS_FACTOR 40

/* writemask number for objects over the chart */
#define CHART_MASK 0x07C0

/* knots to meters conversion */
#define KTS_TO_METERS ((1.0/3600.0)*1856.054) /* hr/sec * m/kt */

/* amount of time between buffer swaps */
#define SMALL_TIME_INC 0.50  /* half of a second */
#define LARGE_TIME_INC 20.0  /* twenty seconds */

/* gains for speed, course, and dive angle dynamic changes */
#define SPEED_CHANGE_GAIN 0.25  /* 0.25 knots per TIME_INC */
#define COURSE_CHANGE_GAIN 1    /* 1 degree per TIME_INC */
#define DIVE_ANGLE_CHANGE_GAIN 1 /* 1 degree per TIME_INC */
```

```
/* 30 by 30 array to hold sonar return data */
#define SONAR_RESOLUTION 30

/* constants for the depth array display */
#define Y_0 5.5
#define Y_5 12.5
#define Y_10 20.5
#define Y_15 28.0
#define Y_20 35.5
#define Y_25 42.5
#define Y_29 48.5

#define X_N_0 3.5
#define X_N_5 11.0
#define X_N_10 18.0
#define X_N_15 25.5
#define X_N_20 33.0
#define X_N_25 40.5
#define X_N_29 46.0

#define X_O_0 52.5
#define X_O_5 60.0
#define X_O_10 67.0
#define X_O_15 74.5
#define X_O_20 82.0
#define X_O_25 89.5
#define X_O_29 95.0

#define Y_BOTTOM 3.0
#define Y_TOP 51.0

#define BASE_NEW_X 5.0
#define BASE_NEW_Y 3.0
#define BASE_OLD_X BASE_NEW_X
#define BASE_OLD_Y 52.0

/* The 3 different scan modes */
#define JUST_SCAN_NO_STORE 0
#define ONE_SCAN_AND_STORE 1
#define COMPLETE_SCAN_AND_STORE 2

#define MAX_PASSES 6

#define MAX_INTEGER 9999999

/* end sonar.h */
```

```
/*+-------------------------------+
 |
 |    disp_arrays.c
 |
 +-------------------------------+              */

/* Displays the depth arrays and performs the grid search regression analysis on successive
   scans */

#include "sonar.h"
#include "gl.h"
#include "math.h"
#include "device.h"

display_depth_arrays(scan_mode, drift, tilt_inc, tilt_angle, scan_complete,
 request_depth_array_display, range_setting, beam_inc, left_depth_array_active,
 left_array_max_depth, left_array_min_depth, right_array_max_depth, right_array_min_depth,
 left_x_coord, left_y_coord, left_depth, right_x_coord, right_y_coord, right_depth)

 short scan_complete, request_depth_array_display, left_depth_array_active, scan_mode;
 int range_setting, tilt_inc, tilt_angle, left_array_max_depth, left_array_min_depth,
  right_array_max_depth, right_array_min_depth;
 float beam_inc, drift, left_x_coord, left_y_coord, left_depth, right_x_coord, right_y_coord,
  right_depth;

{
 extern int bottom_data[BOTTOM_POINTS_HEIGHT][BOTTOM_POINTS_WIDTH];
 extern int left_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
 extern int right_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
 extern int max_depth, min_depth;
 char str[40];
 int i, j;
 int array_depth, e_cog, a_cog, water_depth_at_ave;
 float cog_x, cog_y, a_distance, e_distance, ave_x, ave_y, depth_below_auv;
 Colorindex colour;
 short first_pass, too_many_passes, new_min_criterion, analysis_completed, first_match;
 int number_of_cells, number_of_passes, row_offset, col_offset, cell_location, row_inc, col_inc;
 float criterion_array[8], min_criterion, x_dist, y_dist;
 short right_array_empty, left_array_empty;

 if (request_depth_array_display || scan_complete)
 /* display the depth arrays */
 {
  first_pass = TRUE;

  viewport(0, 1023, 0, 767);
  ortho2(0.0,100.0,0.0,75.0);

  color(BLACK);
  clear();
  swapbuffers();

  while (TRUE)
  {
```

```
if (getbutton(QKEY))
 break;
else
{  /* display the screen */
 color(CYAN);
 clear();
 /* title info first */
 color(BLACK);
 cmov2(25.0,70.0);
 charstr("SONAR RANGE:");
 sprintf(str, "%3d", range_setting);
 cmov2(40.0, 70.0);
 charstr(str);
 cmov2(44.0, 70.0);
 charstr("M");

 cmov2(60.0,70.0);
 charstr("SCALE: 1 - ");
 sprintf(str, "%3.1f", beam_inc);
 cmov2(71.0, 70.0);
 charstr(str);
 cmov2(75.0, 70.0);
 charstr("M");

 cmov2(22.0, 67.0);
 charstr("NEW SCAN");
 cmov2(70.0, 67.0);
 charstr("OLD SCAN");

 cmov2(19.0, 64.0);
 charstr("X_COORD:");
 cmov2(19.0, 62.0);
 charstr("Y_COORD:");
 cmov2(19.0, 60.0);
 charstr("DEPTH:");

 cmov2(65.0, 64.0);
 charstr("X_COORD:");
 cmov2(65.0, 62.0);
 charstr("Y_COORD:");
 cmov2(65.0, 60.0);
 charstr("DEPTH:");

 cmov2(35.0, 56.0);
 charstr("COG:");
 cmov2(51.0, 56.0);
 charstr("DISTANCE:");
 cmov2(68.0, 56.0);
 charstr("M");

 /* max/min headings */
 cmov2(6.0,53.0);
 charstr("MAX DEPTH: ");
 cmov2(28.0,53.0);
```

```
charstr("MIN DEPTH: ");
/* old scan */
cmov2(56.0,53.0);
charstr("MAX DEPTH: ");
cmov2(78.0,53.0);
charstr("MIN DEPTH: ");

/* build the array boxes */
/* first, the numbers along the sides */
/* new scan, left side */
cmov2(1.0, Y_0);
charstr("0");
cmov2(1.0, Y_5);
charstr("5");
cmov2(0.5, Y_10);
charstr("10");
cmov2(0.5, Y_15);
charstr("15");
cmov2(0.5, Y_20);
charstr("20");
cmov2(0.5, Y_25);
charstr("25");
cmov2(0.5, Y_29);
charstr("29");

/* middle of both scans */
cmov2(49.5, Y_0);
charstr("0");
cmov2(49.5, Y_5);
charstr("5");
cmov2(49.0, Y_10);
charstr("10");
cmov2(49.0, Y_15);
charstr("15");
cmov2(49.0, Y_20);
charstr("20");
cmov2(49.0, Y_25);
charstr("25");
cmov2(49.0, Y_29);
charstr("29");

/* right of old scan */
cmov2(98.0, Y_0);
charstr("0");
cmov2(98.0, Y_5);
charstr("5");
cmov2(97.5, Y_10);
charstr("10");
cmov2(97.5, Y_15);
charstr("15");
cmov2(97.5, Y_20);
charstr("20");
cmov2(97.5, Y_25);
charstr("25");
```

```
cmov2(97.5, Y_29);
charstr("29");

/* new scan bottom */
cmov2(X_N_0, Y_BOTTOM);
charstr("0");
cmov2(X_N_5, Y_BOTTOM);
charstr("5");
cmov2(X_N_10, Y_BOTTOM);
charstr("10");
cmov2(X_N_15, Y_BOTTOM);
charstr("15");
cmov2(X_N_20, Y_BOTTOM);
charstr("20");
cmov2(X_N_25, Y_BOTTOM);
charstr("25");
cmov2(X_N_29, Y_BOTTOM);
charstr("29");

/* new scan top */
cmov2(X_N_0, Y_TOP);
charstr("0");
cmov2(X_N_5, Y_TOP);
charstr("5");
cmov2(X_N_10, Y_TOP);
charstr("10");
cmov2(X_N_15, Y_TOP);
charstr("15");
cmov2(X_N_20, Y_TOP);
charstr("20");
cmov2(X_N_25, Y_TOP);
charstr("25");
cmov2(X_N_29, Y_TOP);
charstr("29");

/* old scan bottom */
cmov2(X_O_0, Y_BOTTOM);
charstr("0");
cmov2(X_O_5, Y_BOTTOM);
charstr("5");
cmov2(X_O_10, Y_BOTTOM);
charstr("10");
cmov2(X_O_15, Y_BOTTOM);
charstr("15");
cmov2(X_O_20, Y_BOTTOM);
charstr("20");
cmov2(X_O_25, Y_BOTTOM);
charstr("25");
cmov2(X_O_29, Y_BOTTOM);
charstr("29");

/* old scan top */
cmov2(X_O_0, Y_TOP);
charstr("0");
```

```
cmov2(X_O_5, Y_TOP);
charstr("5");
cmov2(X_O_10, Y_TOP);
charstr("10");
cmov2(X_O_15, Y_TOP);
charstr("15");
cmov2(X_O_20, Y_TOP);
charstr("20");
cmov2(X_O_25, Y_TOP);
charstr("25");
cmov2(X_O_29, Y_TOP);
charstr("29");

/* now display the arrays */
/* first determine active array. Its contents will be displayed on the left (NEW) and the
  inactive array on the right (OLD) */

/* fill backgrounds */
color(RED);
/* new */
rectf(3.0,5.0,48.0,50.0);
/* old */
rectf(52.0,5.0,97.0,50.0);

if (left_depth_array_active)
{ /* left array has new scan info */
 for (i=0; i<SONAR_RESOLUTION; ++i)
 {
  for (j=0; j<SONAR_RESOLUTION; ++j)
  {
   /* first fill in new array */
   array_depth = left_depth_array[i][j];
   if (array_depth != 99999)
   { /* we have contact depth data */
    /* fill in that rectangle */
    colour = (Colorindex)(MAX_COLOR_NUMBER - ((((float)array_depth - min_depth)/
     (max_depth - min_depth)) * NUMBER_OF_COLORS));
    color(colour);
    rectf(BASE_NEW_Y + (j * 1.5), BASE_NEW_X + (i * 1.5), BASE_NEW_Y + ((j + 1) * 1.5),
     BASE_NEW_X + ((i + 1) * 1.5));
   }
   /* next, fill in old array */
   array_depth = right_depth_array[i][j];
   if (array_depth != 99999)
   { /* we have contact depth data */
    /* fill in that rectangle */
    colour = (Colorindex)(MAX_COLOR_NUMBER - ((((float)array_depth - min_depth)/
     (max_depth - min_depth)) * NUMBER_OF_COLORS));
    color(colour);
    rectf(BASE_OLD_Y + (j * 1.5), BASE_OLD_X + (i * 1.5), BASE_OLD_Y + ((j + 1) * 1.5),
     BASE_OLD_X + ((i + 1) * 1.5));
   }
  } /* end for */
 } /* end for */
```

```
/* put up AUV position and MAX,MIN array depths */
color(BLACK);
/* new */
sprintf(str, "%4d", left_array_max_depth);
cmov2(18.0,53.0);
charstr(str);
sprintf(str, "%4d", left_array_min_depth);
cmov2(38.0,53.0);
charstr(str);
/* old */
sprintf(str, "%4d", right_array_max_depth);
cmov2(67.0,53.0);
charstr(str);
sprintf(str, "%4d", right_array_min_depth);
cmov2(88.0,53.0);
charstr(str);

/* display new coord info */
sprintf(str, "%4.1f", left_x_coord);
cmov2(27.0,64.0);
charstr(str);
sprintf(str, "%4.1f", left_y_coord);
cmov2(27.0,62.0);
charstr(str);
sprintf(str, "%4.1f", left_depth);
cmov2(27.0,60.0);
charstr(str);

/* display old coord info */
sprintf(str, "%4.1f", right_x_coord);
cmov2(75.0,64.0);
charstr(str);
sprintf(str, "%4.1f", right_y_coord);
cmov2(75.0,62.0);
charstr(str);
sprintf(str, "%4.1f", right_depth);
cmov2(75.0,60.0);
charstr(str);

/* calc actual cog and distance traveled */
cog_x = left_x_coord - right_x_coord;
cog_y = left_y_coord - right_y_coord;
a_distance = sqrt((cog_x * cog_x)+(cog_y * cog_y));
/* cog */
/* 9 cases */
if ((cog_x == 0) && (cog_y == 0))
 a_cog= 0;
else if ((cog_x == 0) && (cog_y > 0))
 a_cog= 0;
else if ((cog_x == 0) && (cog_y < 0))
 a_cog= 180;
else if ((cog_y == 0) && (cog_x < 0))
 a_cog= 270;
else if ((cog_y == 0) && (cog_x > 0))
```

```
 a_cog= 90;
else if ((cog_x < 0) && (cog_y > 0)) /* 270 - 360 */
 a_cog= 270 + (int)(atan(cog_y/(-1 * cog_x)) * RTOD);
else if ((cog_x < 0) && (cog_y < 0)) /* 180 - 270 */
 a_cog= 270 - (int)(atan(cog_y/cog_x) * RTOD);
else if ((cog_x > 0) && (cog_y > 0)) /* 0 - 90 */
 a_cog= 90 - (int)(atan(cog_y/cog_x) * RTOD);
else if ((cog_x > 0) && (cog_y < 0)) /* 90 - 180 */
 a_cog= 90 + (int)(atan((-1 * cog_y)/cog_x) * RTOD);

sprintf(str, "%3d", a_cog);
cmov2(39.0,56.0);
charstr(str);
sprintf(str, "%4.1f", a_distance);
cmov2(63.0,56.0);
charstr(str);

} /* end if left active */
else /* right array active */
{ /* right array has new scan info */
 for (i=0; i<SONAR_RESOLUTION; ++i)
 {
  for (j=0; j<SONAR_RESOLUTION; ++j)
  {
   /* first fill in new array */
   array_depth = right_depth_array[i][j];
   if (array_depth != 99999)
   { /* we have contact depth data */
    /* fill in that rectangle */
    colour = (Colorindex)(MAX_COLOR_NUMBER - ((((float)array_depth - min_depth)/
     (max_depth - min_depth)) * NUMBER_OF_COLORS));
    color(colour);
    rectf(BASE_NEW_Y + (j * 1.5), BASE_NEW_X + (i * 1.5), BASE_NEW_Y + ((j + 1) * 1.5),
     BASE_NEW_X + ((i + 1) * 1.5));
   }
   /* next, fill in old array */
   array_depth = left_depth_array[i][j];
   if (array_depth != 99999)
   { /* we have contact depth data */
    /* fill in that rectangle */
    colour = (Colorindex)(MAX_COLOR_NUMBER - ((((float)array_depth - min_depth)/
     (max_depth - min_depth)) * NUMBER_OF_COLORS));
    color(colour);
    rectf(BASE_OLD_Y + (j * 1.5), BASE_OLD_X + (i * 1.5), BASE_OLD_Y + ((j + 1) * 1.5),
     BASE_OLD_X + ((i + 1) * 1.5));
   }
  } /* end for */
 } /* end for */

 /* put up AUV position and MAX,MIN array depths */
 color(BLACK);
 /* new */
 sprintf(str, "%4d", right_array_max_depth);
 cmov2(18.0,53.0);
```

```
charstr(str);
sprintf(str, "%4d", right_array_min_depth);
cmov2(38.0,53.0);
charstr(str);
/* old */
sprintf(str, "%4d", left_array_max_depth);
cmov2(67.0,53.0);
charstr(str);
sprintf(str, "%4d", left_array_min_depth);
cmov2(88.0,53.0);
charstr(str);

/* display new coord info */
sprintf(str, "%4.1f", right_x_coord);
cmov2(27.0,64.0);
charstr(str);
sprintf(str, "%4.1f", right_y_coord);
cmov2(27.0,62.0);
charstr(str);
sprintf(str, "%4.1f", right_depth);
cmov2(27.0,60.0);
charstr(str);

/* display old coord info */
sprintf(str, "%4.1f", left_x_coord);
cmov2(75.0,64.0);
charstr(str);
sprintf(str, "%4.1f", left_y_coord);
cmov2(75.0,62.0);
charstr(str);
sprintf(str, "%4.1f", left_depth);
cmov2(75.0,60.0);
charstr(str);

/* calc cog and distance traveled */
cog_x = right_x_coord - left_x_coord;
cog_y = right_y_coord - left_y_coord;
a_distance = sqrt((cog_x * cog_x)+(cog_y * cog_y));
/* cog */
/* 9 cases */
if ((cog_x == 0) && (cog_y == 0))
 a_cog = 0;
else if ((cog_x == 0) && (cog_y > 0))
 a_cog = 0;
else if ((cog_x == 0) && (cog_y < 0))
 a_cog = 180;
else if ((cog_y == 0) && (cog_x < 0))
 a_cog = 270;
else if ((cog_y == 0) && (cog_x > 0))
 a_cog = 90;
else if ((cog_x < 0) && (cog_y > 0)) /* 270 - 360 */
 a_cog = 270 + (int)(atan(cog_y/(-1 * cog_x)) * RTOD);
else if ((cog_x < 0) && (cog_y < 0)) /* 180 - 270 */
 a_cog = 270 - (int)(atan(cog_y/cog_x) * RTOD);
```

```
 else if ((cog_x > 0) && (cog_y > 0)) /* 0 - 90 */
  a_cog = 90 - (int)(atan(cog_y/cog_x) * RTOD);
 else if ((cog_x > 0) && (cog_y < 0)) /* 90 - 180 */
  a_cog = 90 + (int)(atan((-1 * cog_y)/cog_x) * RTOD);

 sprintf(str, "%3d", a_cog);
 cmov2(39.0,56.0);
 charstr(str);
 sprintf(str, "%4.1f", a_distance);
 cmov2(63.0,56.0);
 charstr(str);

 } /* end if right active */

/* now overlay grid & boarders */
color(YELLOW);
linewidth(1);
for (i=0; i<5; ++i)
{
 /* new scan verticles */
 move2(10.5 + (i * 7.5), 5.0);
 draw2(10.5 + (i * 7.5), 50.0);
 /* old scan verticles */
 move2(59.5 + (i * 7.5), 5.0);
 draw2(59.5 + (i * 7.5), 50.0);
 /* new scan horizontals */
 move2(3.0, 12.5 + (i * 7.5));
 draw2(48.0, 12.5 + (i * 7.5));
 /* old scan horizontals */
 move2(52.0, 12.5 + (i * 7.5));
 draw2(97.0, 12.5 + (i * 7.5));
}
/* boarders */
linewidth(2);
color(BLACK);
rect(3.0,5.0,48.0,50.0);
rect(52.0,5.0,97.0,50.0);

swapbuffers();

if (first_pass)
{
 first_match = TRUE;
 number_of_cells = 0;
 number_of_passes = 0;
 too_many_passes = FALSE;
 min_criterion = 0.0;
 row_offset = 0;
 col_offset = 0;
 analysis_completed = FALSE;

 printf("0EW COMPARE ************");

 for (cell_location = 0; cell_location < 8; cell_location = cell_location + 1)
```

```
/* clear all array cells */
criterion_array[cell_location] = 0;

/* calculate criterion function for starting location */
/* both depth arrays exactly overlayed */
for (i = 0; i < 30; ++i)
{
 for (j = 0; j < 30; ++j)
 {
  if ((left_depth_array[i][j] != 99999) && (right_depth_array[i][j] != 99999))
  { /* both cells have data */
   number_of_cells = number_of_cells + 1;
   min_criterion = min_criterion + ((left_depth_array[i][j] - right_depth_array[i][j]) *
    (left_depth_array[i][j] - right_depth_array[i][j]));
  } /* if data */
 } /* inner for */
} /* outer for */

if (number_of_cells == 0)
{ /* no need to continue */
 first_match = FALSE;
 printf(" NO CELLS MATCH ON FIRST ANALYSIS");
}
else
{ /* continue */
 /* normalize the criterion value with the # of cells used */
 min_criterion = min_criterion / number_of_cells;
 printf(" Starting min_criterion = %f",min_criterion);
 printf(" Starting number of cells = %d",number_of_cells);
 /* 15 * 15 * pi = approx 706 total cells available */
 printf(" Percentage of total cells = %f", ((number_of_cells/706.0) * 100.0));

 /* now begin searching for best possible match of both
  depth arrays */

 while (TRUE)
 {
  /* start a new pass, increment pass counter */
  number_of_passes = number_of_passes + 1;
  printf(" NEW PASS PPPPPPPPPPP");
  printf(" Pass number = %d",number_of_passes);

  /* commence calculation of criterion function
   values for all 8 outer cells in order */
  for (cell_location = 0; cell_location < 8;
   cell_location = cell_location + 1)
  {
   printf(" NEW CELL CCCCCCC");
   printf(" Cell_location = %d", cell_location);
   if (cell_location == 0)
   {
    row_offset = row_offset + -1;
    col_offset = col_offset + -1;
   }
```

```
else if (cell_location == 1)
{
 row_offset = row_offset - 1;
}
else if (cell_location == 2)
{
 row_offset = row_offset - 1;
 col_offset = col_offset + 1;
}
else if (cell_location == 3)
{
 col_offset = col_offset - 1;
}
else if (cell_location == 4)
{
 col_offset = col_offset + 1;
}
else it (cell_location == 5)
{
 row_offset = row_offset + 1;
 col_offset = col_offset - 1;
}
else if (cell_location == 6)
{
 row_offset = row_offset + 1;
}
else if (cell_location == 7)
{ /* cell_location == 7 */
 row_offset = row_offset + 1;
 col_offset = col_offset + 1;
}

/* we now have new cell_location (row/col position) */
/* calculate criterion function for the new cell */
/* reset number of cells and array cell at that
 cell location to zero */
number_of_cells = 0;
criterion_array[cell_location] = 0;

for (i=0; i<30; ++i)
{
 for (j=0; j<30; ++j)
 {
  if (((i+row_offset) > -1) && ((i+row_offset) < 30) && ((j+col_offset) > -1) &&
   ((j+col_offset) < 30))
  { /* we are in array bounds */
   if (left_depth_array_active)
   {
    if ((left_depth_array[i+row_offset][j+col_offset] != 99999) &&
     (right_depth_array[i][j] != 99999))
    { /* both cells have data */
     number_of_cells = number_of_cells + 1;
     criterion_array[cell_location] = criterion_array[cell_location] +
      ((left_depth_array[i+row_offset] [j+col_offset] - right_depth_array[i][j]) *
```

```
    (left_depth_array[i+row_offset][j+col_offset] - right_depth_array[i][j]));
   }
  } /* end if left active */
  else
  { /* right active */
   if ((right_depth_array[i+row_offset][j+col_offset] != 99999) &&
    (left_depth_array[i][j] != 99999))
    { /* both cells have data */
     number_of_cells = number_of_cells + 1;
     criterion_array[cell_location] = criterion_array[cell_location] +
      ((right_depth_array[i+row_offset][j+col_offset] - left_depth_array[i][j]) *
      (right_depth_array[i+row_offset][j+col_offset] - left_depth_array[i][j]));
    }
  } /* end else right active */
 } /* end if in array bounds */
} /* end inner for */
} /* end outer for */

printf(" number_of_cells = %d", number_of_cells);

if (number_of_cells == 0)
 criterion_array[cell_location] = MAX_INTEGER;
else
{ /* normalize that criterion function value */
 criterion_array[cell_location] = criterion_array[cell_location]/number_of_cells;
}

printf(" Criterion Value = %f",criterion_array[cell_location]);
/* reset row and col offsets back to starting positions before offsetting to next cell */
if (cell_location == 0)
{
 row_offset = row_offset + 1;
 col_offset = col_offset + 1;
}
else if (cell_location == 1)
{
 row_offset = row_offset + 1;
}
else if (cell_location == 2)
{
 row_offset = row_offset + 1;
 col_offset = col_offset - 1;
}
else if (cell_location == 3)
{
 col_offset = col_offset + 1;
}
else if (cell_location == 4)
{
 col_offset = col_offset - 1;
}
else if (cell_location == 5)
{
 row_offset = row_offset - 1;
```

```
 col_offset = col_offset + 1;
 }
 else if (cell_location == 6)
 {
 row_offset = row_offset - 1;
 }
 else if (cell_location == 7)
 {
 row_offset = row_offset - 1;
 col_offset = col_offset - 1;
 }

 } /* end for cell locations 0 - 8 */

 /* we now have criterion function values for all surrounding cells */
 /* determine which cell holds the smallest value */
 /* reset row/col inc */
 row_inc = 0;
 col_inc = 0;

 /* assume no movement */
 new_min_criterion = FALSE;

 for (cell_location = 0; cell_location < 8;
  cell_location = cell_location + 1)
 {
  if (criterion_array[cell_location] <
   min_criterion)
  { /* we have a new criterion minimum */
   min_criterion = criterion_array[cell_location];
   new_min_criterion = TRUE;

   switch (cell_location)
   {
    case 0: row_inc = -1;
     col_inc = -1;
     break;
    case 1: row_inc = -1;
     col_inc = 0;
     break;
    case 2: row_inc = -1;
     col_inc = 1;
     break;
    case 3: col_inc = -1;
     row_inc = 0;
     break;
    case 4: col_inc = 1;
     row_inc = 0;
     break;
    case 5: row_inc = 1;
     col_inc = -1;
     break;
    case 6: row_inc = 1;
     col_inc = 0;
```

```
   break;
  case 7: row_inc = 1;
   col_inc = 1;
   break;
  } /* switch */
 } /* end if new criterion */
} /* end for all cell locations */

/* we now have minimum value of all 9 cells */
/* now see if new min_criterion is not center cell */
if (new_min_criterion)
{ /* we moved */
 row_offset = row_offset + row_inc;
 col_offset = col_offset + col_inc;
 if (number_of_passes > MAX_PASSES)
 {
  too_many_passes = TRUE;
  analysis_completed = TRUE;
 }
} /* end if new min */
else /* at best match - no movement */
 analysis_completed = TRUE;

/* print out info */
if (too_many_passes)
 printf(" REACHED MAXIMUM NUMBER OF PASSES!");

printf(" row_offset = %d", row_offset);
printf(" col_offset = %d", col_offset);

/* determine distance moved */
y_dist = row_offset * beam_inc * -1;
x_dist = col_offset * beam_inc * -1;
printf(" x_dist = %f", x_dist);
printf(" y_dist = %f", y_dist);

/* calc estimated dog & cog */
/* 9 cases */
if ((x_dist == 0) && (y_dist == 0))
 e_cog= 0;
else if ((x_dist == 0) && (y_dist > 0))
 e_cog= 0;
else if ((x_dist == 0) && (y_dist < 0))
 e_cog= 180;
else if ((y_dist == 0) && (x_dist < 0))
 e_cog= 270;
else if ((y_dist == 0) && (x_dist > 0))
 e_cog= 90;
else if ((x_dist < 0) && (y_dist > 0)) /* 270 - 360 */
 e_cog= 270 + (int)(atan(y_dist/(-1 * x_dist)) * RTOD);
else if ((x_dist < 0) && (y_dist < 0)) /* 180 - 270 */
 e_cog= 270 - (int)(atan(y_dist/x_dist) * RTOD);
else if ((x_dist > 0) && (y_dist > 0)) /* 0 - 90 */
 e_cog= 90 - (int)(atan(y_dist/x_dist) * RTOD);
```

```
else if ((x_dist > 0) && (y_dist < 0)) /* 90 - 180 */
 e_cog= 90 + (int)(atan((-1 * y_dist)/x_dist) * RTOD);
printf(" ESTIMATED COG = %d", e_cog);

/* distance over ground */
e_distance = sqrt((x_dist * x_dist) + (y_dist * y_dist));
printf(" ESTIMATED DISTANCE = %f", e_distance);

if (analysis_completed)
 break;

} /* end while TRUE */

printf("0OMPARE COMPLETE ########");
printf("0umber of moves = %d", number_of_passes - 1);
printf(" ACTUAL COG = %d degrees", a_cog);
printf(" FINAL ESTIMATED COG = %d", e_cog);
printf(" ACTUAL DISTANCE = %f meters", a_distance);
printf(" FINAL ESTIMATED DISTANCE = %f", e_distance);

/* calc number of moves horizontally and vertically */
/* actual verticle/ horizontal */
printf(" Actual horiz. grid displacement = %d", round(sin(a_cog * DTOR) *
 a_distance / beam_inc));
printf(" Actual vert. grid displacement = %d", round(cos(a_cog * DTOR) *
 a_distance / beam_inc));

/* estimated verticle/ horizontal */
printf(" Estimated horiz. grid displacement = %d", (int)(x_dist / beam_inc));
printf(" Estimated vert. grid displacement = %d", (int)(y_dist / beam_inc));

if (left_depth_array_active)
{
 printf(" Starting x_coord = %f", right_x_coord);
 printf(" Starting y_coord = %f", right_y_coord);
 printf(" Ending x_coord = %f", left_x_coord);
 printf(" Ending y_coord = %f", left_y_coord);
}
else /* right active */
{
 printf(" Starting x_coord = %f", left_x_coord);
 printf(" Starting y_coord = %f", left_y_coord);
 printf(" Ending x_coord = %f", right_x_coord);
 printf(" Ending y_coord = %f", right_y_coord);
}

printf(" Drift = %f knots", drift);
/* calculate the average depth below auv here */
/* first average the x and y coords */
ave_x = (right_x_coord + left_x_coord)/2;
ave_y = (right_y_coord + left_y_coord)/2;
/* convert these to array indices */
water_depth_at_ave = bottom_data[(int)(ave_y/RESOLUTION_IN_METERS)]
 [(int)(ave_x/RESOLUTION_IN_METERS)];
```

```
depth_below_auv = (water_depth_at_ave * M_PER_FEET) - ((right_depth + left_depth)/2);
printf(" Average depth below auv = %f meters", depth_below_auv);
/* calc percentage of depth below auv with respect to range_setting */
printf(" Depth below auv/Range Setting = %f percent", (depth_below_auv/range_setting)*
100.0);

printf(" Range Setting = %d meters", range_setting);
printf(" Beam Increment = %f meters", beam_inc);

switch (scan_mode)
{
 case ONE_SCAN_AND_STORE:
  printf(" Scan Mode = ONE SCAN");
  printf(" Tilt Angle = %d degrees", tilt_angle);
  break;

 case COMPLETE_SCAN_AND_STORE:
  printf(" Scan Mode = COMPLETE SCAN");
  printf(" Tilt Increment = %d degrees", tilt_inc);
  break;

 } /* end switch */

 } /* end else continue, first match = TRUE */

 first_pass = FALSE;

 } /* end if first pass */

 } /* end else not exit */

 } /* end while TRUE */

 } /* end if display requested */

} /* end display arrays */
```

```
/*+-------------------------------+
 |                               |
 |     draw_numbers.c            |
 |                               |
 +-------------------------------+              */

/* This procedure is called by make_video to draw large numbers on the video screen for depth,
   range_setting, current range ring location and tilt angle */

#include "sonar.h"
#include "gl.h"

draw_numbers(digit, starting_x, starting_y)

short digit;
float starting_x; /* starting x location of letters */
float starting_y; /* starting y location of letters */

{
color(WHITE);
linewidth(2);
switch (digit)
 {
 case 1:  move2(starting_x, starting_y);
    draw2(starting_x + 3.8, starting_y);
    move2(starting_x + 2.0, starting_y);
    draw2(starting_x + 2.0, starting_y + 4.0);
    break;

 case 2:  move2(starting_x + 3.8, starting_y);
    draw2(starting_x, starting_y);
    draw2(starting_x, starting_y + 2.5);
    draw2(starting_x + 3.8, starting_y + 2.5);
    draw2(starting_x + 3.8, starting_y + 4.0);
    draw2(starting_x, starting_y + 4.0);
    break;

 case 3:  move2(starting_x, starting_y);
    draw2(starting_x + 3.8, starting_y);
    draw2(starting_x + 3.8, starting_y + 4.0);
    draw2(starting_x, starting_y + 4.0);
    move2(starting_x + 1.5, starting_y + 2.5);
    draw2(starting_x + 3.8, starting_y + 2.5);
    break;

 case 4:  move2(starting_x, starting_y + 4.0);
    draw2(starting_x, starting_y + 1.5);
    draw2(starting_x + 3.8, starting_y + 1.5);
    move2(starting_x + 2.5, starting_y);
    draw2(starting_x + 2.5, starting_y + 4.0);
    break;

 case 5:  move2(starting_x, starting_y);
    draw2(starting_x + 3.8, starting_y);
```

```
    draw2(starting_x + 3.8, starting_y + 2.5);
    draw2(starting_x, starting_y + 2.5);
    draw2(starting_x, starting_y + 4.0);
    draw2(starting_x + 3.8, starting_y + 4.0);
    break;

  case 6:  move2(starting_x, starting_y);
    draw2(starting_x + 3.8, starting_y);
    draw2(starting_x + 3.8, starting_y + 2.5);
    draw2(starting_x, starting_y + 2.5);
    move2(starting_x, starting_y);
    draw2(starting_x, starting_y + 4.0);
    draw2(starting_x + 3.8, starting_y + 4.0);
    break;

  case 7:  move2(starting_x + 0.5, starting_y);
    draw2(starting_x + 3.8, starting_y + 4.0);
    move2(starting_x, starting_y + 4.0);
    draw2(starting_x + 3.8, starting_y + 4.0);
    break;

  case 8:  rect(starting_x, starting_y, starting_x + 3.8, starting_y + 2.5);
    rect(starting_x + 0.5, starting_y + 2.5, starting_x + 3.5, starting_y + 4.0);
    break;

  case 9:  rect(starting_x, starting_y + 2.5, starting_x + 3.8, starting_y + 4.0);
    move2(starting_x + 3.8, starting_y + 4.0);
    draw2(starting_x + 3.8, starting_y);
    break;

  case 0:  rect(starting_x, starting_y, starting_x + 3.8, starting_y + 4.0);
    break;


 } /* close switch */

} /* draw_numbers */
```

```
/*+-------------------------------+
 |
 |       edit_dials.c
 |
 +-------------------------------+              */

/* This procedure is called by main_sonar to edit the dial objects depending on how dials
   have changed */

#include "sonar.h"
#include "gl.h"

edit_dials(dials, power_dial_location,volume_dial_location, sector_dial_location,
 range_dial_location, power_dial_change, volume_dial_change, sector_dial_change,
 range_dial_change, volume_tag, power_tag, sector_tag, range_tag)

Object dials; /* object with all dials */
Tag volume_tag, power_tag, sector_tag,
 range_tag;  /* tags for each dial's rotation */
int power_dial_location; /* angular location of dial positions */
int volume_dial_location, sector_dial_location, range_dial_location;
short power_dial_change; /* boolean for dial changes */
short volume_dial_change, sector_dial_change, range_dial_change;

{
 if (power_dial_change)
 { /* edit power dial */
  editobj(dials);
   objreplace(power_tag);
   rotate(-power_dial_location*10, 'z');
  closeobj();
 }

 if (volume_dial_change)
 { /* edit volume dial */
  editobj(dials);
   objreplace(volume_tag);
   rotate(-volume_dial_location*10, 'z');
  closeobj();
 }

 if (sector_dial_change)
 { /* edit sector dial */
  editobj(dials);
   objreplace(sector_tag);
   rotate(sector_dial_location*10, 'z');
  closeobj();
 }

 if (range_dial_change)
 { /* edit range dial */
  editobj(dials);
   objreplace(range_tag);
   rotate(range_dial_location*10, 'z');
```

```
 closeobj();
 }
} /* edit_dials */
```

```
/*+------------------------------+
 |
 |     edit_mask.c
 |
 +------------------------------+              */

/* This procedure is called by main_sonar to edit the scan mask depending on the sector
   dial location */

#include "sonar.h"
#include "gl.h"

edit_scan_mask(scan_mask, scan_heading, scan_heading_change, sector_setting_change,
 sector_setting, scan_mask_tag)

Object scan_mask; /* object masking the video screen */
Tag scan_mask_tag; /* tag for arcf */
int scan_heading, sector_setting;
short scan_heading_change, sector_setting_change;

{
Angle sector_start; /* arcf starting location */
Angle sector_end; /* arcf ending location */

 /* if sector setting or scan heading change, edit the mask */
 if (sector_setting_change || scan_heading_change)
 {
  if (sector_setting==360)
  { /* reset to no scan mask */
   editobj(scan_mask);
    objreplace(scan_mask_tag);
    arcf(0.0,0.0,SONAR_RADIUS,0.0,0.0);
   closeobj();
  }
  else
  { /* other that 360 */
    /* figure new sector_start-ends */

   sector_start=scan_heading-(sector_setting/2);
   if (sector_start < 0)
    sector_start = 360 + sector_start;
   sector_end=(sector_setting/2)+scan_heading;
   if (sector_end > 360)
    sector_end = sector_end - 360;

   /* convert to arc angles */

   sector_start = (360 - sector_start);
   sector_end = (360 - sector_end);

   editobj(scan_mask);
    objreplace(scan_mask_tag);
    arcf(0.0,0.0,SONAR_RADIUS,sector_start * 10,sector_end * 10);
   closeobj();
```

```
  } /* other that 360 */
 } /* if a change */
} /* edit_scan_mask */
```

```
/*+-------------------------------+
 |
 |    edit_scline.c
 |
 +-------------------------------+              */

/* This procedure is called by main_sonar to edit the scan heading line  object */

#include "sonar.h"
#include "gl.h"

edit_scan_heading_line(scan_heading_line, scan_heading, scan_heading_tag)

Object scan_heading_line; /* object for scan heading line */
Tag scan_heading_tag; /* tag for scan heading rotation */
short scan_heading; /* current location */
{

 editobj(scan_heading_line);
  objreplace(scan_heading_tag);
  rotate(scan_heading*-10, 'z');
 closeobj();

} /* edit_scan_heading_line  */
```

```
/*+-------------------------------+
 |
 |      edit_sweep.c
 |
 +-------------------------------+              */

/* This procedure is called by main_sonar to edit the sonar sweep object */

#include "sonar.h"
#include "math.h"
#include "gl.h"

edit_sonar_sweep(sonar_sweep, sweep_clockwise, sonar_sweep_location, sector_sweep_start,
 sector_sweep_end, sector_setting_change, sector_setting, scan_heading,
 scan_heading_change, sonar_sweep_tag, power_on, hoist_down)

Object sonar_sweep;
Tag sonar_sweep_tag;
short *sweep_clockwise, sector_setting_change, scan_heading_change;
int sector_setting, scan_heading, *sonar_sweep_location, *sector_sweep_start, *sector_sweep_end;
short power_on, hoist_down;

{

 if (sector_setting_change || scan_heading_change)
 {

  if (sector_setting != 360) /* get sector start/end values */
  {
   *sector_sweep_start = scan_heading - (sector_setting / 2);
   *sector_sweep_end = scan_heading + (sector_setting / 2);
   if (*sector_sweep_start <= 0)
    *sector_sweep_start = 360 + *sector_sweep_start;
   if (*sector_sweep_end > 360)
    *sector_sweep_end = *sector_sweep_end - 360;
  }
 }  /* sector change or scan heading change  */

 if (sector_setting == 360)
 {
  /* reset sweep direction to clockwise if sector setting = 360 */
  *sweep_clockwise = TRUE;

 }

 /* edit sonar_sweep object */
 if (*sweep_clockwise)
 {
  editobj(sonar_sweep);
   objreplace(sonar_sweep_tag);
   rotate(*sonar_sweep_location * -10, 'z');
  closeobj();
 }
 else /* counter clockwise */
```

```
{
 if (*sonar_sweep_location <= 0)
  *sonar_sweep_location = 360 + *sonar_sweep_location;

 editobj(sonar_sweep);
  objreplace(sonar_sweep_tag);
  rotate((*sonar_sweep_location-24) * -10, 'z');
 closeobj();
}

if (sector_setting != 360)
{ /* check sweep bounds */
 if (*sweep_clockwise &&
  (*sonar_sweep_location < *sector_sweep_end+SWEEP_STEP)&&(*sonar_sweep_location >=
   *sector_sweep_end))
 {
   *sweep_clockwise = FALSE;
 }
 if (!*sweep_clockwise && (*sonar_sweep_location > *sector_sweep_start - SWEEP_STEP) &&
  (*sonar_sweep_location <= *sector_sweep_start))
 {
   *sweep_clockwise = TRUE;
  }
}

if (power_on && hoist_down)
{ /* increment sweep */
 if (*sweep_clockwise)
 {
  *sonar_sweep_location += SWEEP_STEP;
  if (*sonar_sweep_location == 360)
   *sonar_sweep_location = 0;
 }
 else /* counter clockwise , decrement sweep */
 {
  *sonar_sweep_location -= SWEEP_STEP;
  if (*sonar_sweep_location == -8)
   *sonar_sweep_location = 352;
 }
} /* if power on */

}  /* edit_sonar_sweep */
```

```
/*+-------------------------------+
 |
 |      edit_tilt.c
 |
 +-------------------------------+              */

/* This procedure is called by main_sonar to edit the tilt angle indicator in the upper
   left hand corner of the video */

#include "sonar.h"
#include "gl.h"

edit_tilt_angle_indicator(tilt_angle_indicator, tilt_angle_change, tilt_angle, tilt_angle_tag)

Object tilt_angle_indicator;
Tag tilt_angle_tag;
int tilt_angle;
short *tilt_angle_change;

{
 if (*tilt_angle_change)
 {
  editobj(tilt_angle_indicator);
   objreplace(tilt_angle_tag);
   rotate(tilt_angle*10,'z');
  closeobj();
  *tilt_angle_change = FALSE;
 }

} /* edit_tilt_angle_indicator */
```

```
/*+-------------------------------+
 |
 |     edit_toggles.c
 |
 +-------------------------------+              */

/* edit_toggles - this procedure is called by make_sonar_panel to
   build the toggles on the sonar panel */

#include "gl.h"
#include "sonar.h"

edit_toggle(which_toggle, up)
short which_toggle; /* which toggle of the three */
short up; /* up 1, down 0 */
{

Coord start_x;

 switch (which_toggle) {
  case 1:  /* tilt toggle */
   start_x = 15.5;
   break;

  case 2: /* scan direction toggle */
   start_x = 23.5;
   break;

  case 3:  /* range ring toggle */
   start_x = 31.5;
   break;
 } /* end switch */

 color(BLACK);

 if (up)
 {
  rectf(start_x,START_TOGGLE_Y, start_x+1.0,START_TOGGLE_Y + 2.5);
 }

 else
 {
  rectf(start_x,START_TOGGLE_Y-2.5, start_x+1.0,START_TOGGLE_Y);
 }

} /* edit_toggles */
```

```
/*+-------------------------------+
 |
 |      get_posit.c
 |
 +-------------------------------+              */

/* This procedure is called by main_sonar to get the next position of the auv.  This procedure
   contains the auv dynamics. */

#include "gl.h"             /* graphics lib defs */
#include "sonar.h"             /* sonar constants */
#include "math.h"           /* math definitions */

get_next_position(scan_mode, x_coord, y_coord, depth, cog, sog, course, speed, dive_angle,
 selected_course, selected_speed, selected_dive_angle, set, drift, aground_flag, scan_complete)

 int *course, selected_course, *dive_angle, selected_dive_angle, *cog, set;
 float *x_coord, *y_coord, *depth, *speed, selected_speed, *sog, drift;
 short scan_mode, *aground_flag, scan_complete;

{
 extern int bottom_data[BOTTOM_POINTS_WIDTH][BOTTOM_POINTS_HEIGHT];
 static short auv_change = TRUE;/* boolean for auv or current change */
 static int last_set = 0;
 static float last_drift = 0.0;
 /* vars to calculate cog and sog */
 float course_x, course_y, current_x, current_y, cog_x, cog_y;

 if ((scan_mode == JUST_SCAN_NO_STORE) || scan_complete)
 { /* a sonar scan has completed, update config and position */
  /* AUV does not move until first scan is completed if mode
   is other that JUST_SCAN */

  if (selected_course != *course) /* course needs changing */
  {
   auv_change = TRUE;
   /* determine which way is better to turn to reach selected course fastest */
   if (selected_course > 180)
   {
    if ((selected_course > *course) && (*course >= (selected_course - 180)))
    {
     *course = *course + COURSE_CHANGE_GAIN;
    }
    else
    {
     *course = *course - COURSE_CHANGE_GAIN;
    }
   }
   else
   {
    if ((selected_course < *course) && (*course <= (selected_course + 180)))
    {
     *course = *course - COURSE_CHANGE_GAIN;
    }
```

```
  else
  {
   *course = *course + COURSE_CHANGE_GAIN;
  }
 }
 if (*course == 360)
  *course = 0;
 if (*course == -1)
  *course = 359;

 } /* if course needs changing */

 if (selected_speed != *speed) /* speed needs changing */
 {
  auv_change = TRUE;
  if (selected_speed > *speed)
   *speed = *speed + SPEED_CHANGE_GAIN;
  else
   *speed = *speed - SPEED_CHANGE_GAIN;
 }
 if ((*speed > (selected_speed - SPEED_CHANGE_GAIN)) &&
  (*speed < (selected_speed + SPEED_CHANGE_GAIN))) *speed = selected_speed;

 if (selected_dive_angle != *dive_angle) /* dive needs changing */
 {
  auv_change = TRUE;
  if (selected_dive_angle > *dive_angle)
   *dive_angle = *dive_angle + DIVE_ANGLE_CHANGE_GAIN;
  else
   *dive_angle = *dive_angle - DIVE_ANGLE_CHANGE_GAIN;
 }

 if (last_set != set)
 {
  auv_change = TRUE;
  last_set = set;
 }

 if (last_drift != drift)
 {
  auv_change = TRUE;
  last_drift = drift;
 }

 if (auv_change)  /* calc new cog/sog */
 {
 /* calculate cog/sog */
  course_x = sin(*course * DTOR) * *speed;
  course_y = cos(*course * DTOR) * *speed;
  current_x = sin(set * DTOR) * drift;
  current_y = cos(set * DTOR) * drift;
  cog_x = course_x + current_x;
  cog_y = course_y + current_y;
```

```
/* 9 cases */
if ((cog_x == 0) && (cog_y == 0))
 *cog = *course;
else if ((cog_x == 0) && (cog_y > 0))
 *cog = 0;
else if ((cog_x == 0) && (cog_y < 0))
 *cog = 180;
else if ((cog_y == 0) && (cog_x < 0))
 *cog = 270;
else if ((cog_y == 0) && (cog_x > 0))
 *cog = 90;
else if ((cog_x < 0) && (cog_y > 0)) /* 270 - 360 */
 *cog = 270 + (int)(atan(cog_y/(-1 * cog_x)) * RTOD);
else if ((cog_x < 0) && (cog_y < 0)) /* 180 - 270 */
 *cog = 270 - (int)(atan(cog_y/cog_x) * RTOD);
else if ((cog_x > 0) && (cog_y > 0)) /* 0 - 90 */
 *cog = 90 - (int)(atan(cog_y/cog_x) * RTOD);
else if ((cog_x > 0) && (cog_y < 0)) /* 90 - 180 */
 *cog = 90 + (int)(atan((-1 * cog_y)/cog_x) * RTOD);

/* calculate sog */
*sog = sqrt((cog_x * cog_x) + (cog_y * cog_y));

} /* if auv_change */


/* calculate the new x_coord, y_coord, and depth */
if (scan_mode != JUST_SCAN_NO_STORE)
{ /* use large time inc to get large movement */
 *x_coord = *x_coord + (*sog * KTS_TO_METERS * LARGE_TIME_INC *
  cos(*dive_angle*DTOR) * cos((*cog - 90)* DTOR));

 *y_coord = *y_coord + (*sog * KTS_TO_METERS * LARGE_TIME_INC *
  cos(*dive_angle*DTOR) * sin((*cog - 90)* DTOR));

 *depth = *depth + (*sog * KTS_TO_METERS * LARGE_TIME_INC *
  sin(*dive_angle*DTOR));
}
else /* in JUST_SCAN mode, use small time inc */
{
 *x_coord = *x_coord + (*sog * KTS_TO_METERS * SMALL_TIME_INC *
  cos(*dive_angle*DTOR) * cos((*cog - 90)* DTOR));

 *y_coord = *y_coord + (*sog * KTS_TO_METERS * SMALL_TIME_INC *
  cos(*dive_angle*DTOR) * sin((*cog - 90)* DTOR));

 *depth = *depth + (*sog * KTS_TO_METERS * SMALL_TIME_INC *
  sin(*dive_angle*DTOR));
}

/* assume AUV not aground and in operating area */
*aground_flag = FALSE;

/* check to see if AUV is in operating area */
```

```
if ((*x_coord < 0.0) || (*x_coord > 1000.0) || (*y_coord > 0.0) || (*y_coord < -1000.0))
 /* y_coord is a negative number */
 *aground_flag = TRUE;

/* check to see if AUV is aground */
if ((FEET_PER_M * *depth) >= bottom_data[(int)((-1 * *y_coord)/
 RESOLUTION_IN_METERS)][(int)(*x_coord/RESOLUTION_IN_METERS)])
 *aground_flag = TRUE;

} /* end if scan_complete or JUST_SCAN mode */

auv_change = FALSE; /* reset, assume no change next time */

} /* end get posit */
```

```
/*+------------------------------+
 |
 |      get_sets.c
 |
 +------------------------------+              */

/* This procedure is called by main_sonar to get the values from the operator's controls
   (mouse and dials) when a change in location/depth is requested */

#include "gl.h"              /* graphics lib defs */
#include "sonar.h"            /* sonar constants */

get_auv_settings(scan_mode, x_coord, y_coord, depth, course, speed, dive_angle, selected_course,
 selected_speed, selected_dive_angle, set, drift, cog, sog, chart, aground_flag, tilt_inc)

 int *course, *dive_angle, *set, *cog, *selected_course, *selected_dive_angle, *tilt_inc;
 float *x_coord, *y_coord, *speed, *depth, *drift, *sog, *selected_speed;
 Object chart;
 short *scan_mode, *aground_flag;

 {
 Colorindex wmask;
 short done=FALSE;

 /* enable bit planes so we can switch between writemasks */
 wmask=(1<<getplanes()) -1;
 writemask(wmask);

 /* build screen */
 /* make auv instrument readout */
 make_readout(*scan_mode, *x_coord, *y_coord, *depth, *course, *speed, *dive_angle,
  *selected_course, *selected_speed, *selected_dive_angle, *set, *drift, *cog, *sog,
  *tilt_inc);
 /* make the depth indicator bar */
 make_depth_bar(*x_coord, *y_coord, *depth);
 /* make the instruction billboard */
 make_instructions(*aground_flag);

 callobj(chart);

 swapbuffers();

 callobj(chart);

 while(TRUE)
 {
  /* read the instrument panel */
  read_settings(scan_mode, x_coord, y_coord, depth, course, speed, dive_angle, selected_course,
   selected_speed, selected_dive_angle, set, drift, cog, sog, tilt_inc, &done);
  if (done)
   break;
  else
  {
   /* change the writemask so the chart does not have to be rebuilt */
```

```
 writemask(CHART_MASK);
 /* indicate the position of the auv on the chart */
 make_position_plus(*x_coord, *y_coord);
 /* switch back the writemask */
 writemask(wmask);

 /* make the auv instrument readout */
 make_readout(*scan_mode, *x_coord, *y_coord, *depth, *course, *speed, *dive_angle,
  *selected_course, *selected_speed, *selected_dive_angle, *set, *drift, *cog, *sog,
  *tilt_inc);

 /* make the depth indicator bar */
 make_depth_bar(*x_coord, *y_coord, *depth);

 /* make the instruction billboard */
 make_instructions(*aground_flag);

 swapbuffers();

 } /* else not done */

} /* while */

/* reset aground flag */
*aground_flag = FALSE;

} /* get_auv_settings */
```

```
/*+-------------------------------+
 |
 |     init_arrays.c
 |
 +-------------------------------+              */

/* Initializes the depth arrays and max/min values */

#include "sonar.h"
#include "gl.h"

initialize_depth_arrays(sweep_location, left_depth_array_active, left_array_max_depth,
 left_array_min_depth, right_array_max_depth, right_array_min_depth, x_coord, y_coord, depth,
 left_x_coord, left_y_coord, left_depth, right_x_coord, right_y_coord, right_depth)

 short *left_depth_array_active;
 int *sweep_location, *left_array_max_depth, *left_array_min_depth, *right_array_max_depth,
  *right_array_min_depth;
 float x_coord, y_coord, depth, *left_x_coord, *left_y_coord, *left_depth, *right_x_coord,
  *right_y_coord, *right_depth;

 {
 extern int left_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
 extern int right_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
 int i, j;

 for (i=0; i<SONAR_RESOLUTION; ++i)
 {
  for (j=0; j<SONAR_RESOLUTION; ++j)
  {
   right_depth_array[i][j] = 99999;
   left_depth_array[i][j] = 99999;
  } /* inner for */
 } /* outer for */

 *right_array_min_depth = 99999;
 *right_array_max_depth = -99999;
 *left_array_min_depth = 99999;
 *left_array_max_depth = -99999;
 *left_depth_array_active = TRUE;
 *sweep_location = 0;
 *left_x_coord = x_coord;
 *left_y_coord = -1 * y_coord;
 *left_depth = depth;
 *right_x_coord = x_coord;
 *right_y_coord = -1 * y_coord;
 *right_depth = depth;

 } /* end initialize_depth_arrays */
```

```
/*+------------------------------+
 |
 |       init_iris.c
 |
 +------------------------------+              */

/* This procedure is called by main_sonar to initialize the graphics environment for the iris
   workstation */

#include "gl.h"            /* graphics definitions  */
#include "device.h"          /* device definitions  */
#include "sonar.h"           /* sonar constants  */

init_iris()

{
Colorindex wmask;

/* build a cursonr that looks like first draft AUV */
static unsigned short cursordef[16] ={ 0x3FF8,0x2108,0x2D68,0x2108,
 0x3D78,0x2548,0x2548,0x2448,
 0x27C8,0x2008,0xE00E,0xE00E,
 0xE00E,0x2008,0x2008,0x1FF0};

int i; /* loop control */

ginit();               /* initialize the IRIS system */
doublebuffer();          /* put the IRIS into double buffer mode */
gconfig();             /* (means use the above command settings) */
/* enable bitplanes  */
wmask=(1<<getplanes()) -1;
writemask(wmask);

/* exit key */
qdevice(EKEY); /* exit program key */
qdevice(QKEY); /* continue scanning key */
qdevice(SKEY); /* stop scanning key */
qdevice(ZEROKEY); /* set to just scan mode key */
qdevice(ONEKEY);  /* set to one sweep and display mode key */
qdevice(TWOKEY);  /* set to complete scan and display mode key */
qdevice(UPARROWKEY); /* increase tilt increment key */
qdevice(DOWNARROWKEY); /* decrease tilt increment key */

/* define my cursor that looks like a sub  */
defcursor(MY_CURSOR,cursordef);
setcursor(MY_CURSOR, CURSOR_COLOR, wmask);
attachcursor(MOUSEX, MOUSEY);
curson();

/* set noise values */
noise(MOUSEX,3);
noise(MOUSEY,3);
noise(DIAL0,2);
noise(DIAL1,2);
```

```
noise(DIAL2,2);
noise(DIAL3,2);
noise(DIAL4,6);
noise(DIAL5,6);
noise(DIAL6,6);

/* set valuators */
setvaluator(MOUSEX,0,0,1023);
setvaluator(MOUSEY,0,0,767);
setvaluator(DIAL0,0,0,90);   /* power */
setvaluator(DIAL1,0,0,360);  /* volume/hoist */
setvaluator(DIAL2,0,0,280);  /* sector */
setvaluator(DIAL3,0,0,300);  /* range */
setvaluator(DIAL4,0,0,360);  /* course */
setvaluator(DIAL5,0,0,360);  /* set */
setvaluator(DIAL6,0,-90,90); /* dive_angle */


/* set up colors for sonar video */
mapcolor(GREY,120,120,120);
mapcolor(ORANGE,255,131,0);

/* colors for ocean bottom chart */
for (i=0; i<24; ++i)
 mapcolor(i+16,0,0,i*11);   /* dark blue to blue (16 - 39)*/
for (i=0; i<24; ++i)
 mapcolor(i+40,i*11,i*11,255);   /* light blue to white (40-63) */

/* colors for chartmask arrows */
for (i=64; i<128; ++i)
 mapcolor(i,0,0,0);   /* black (64-127) */
for (i=128; i<256; ++i)
 mapcolor(i,255,0,0);   /* red (128-255) */
for (i=256; i<512; ++i)
 mapcolor(i,0,255,0);   /* green (256-511) */
for (i=512; i<1024; ++i)
 mapcolor(i,255,255,0);   /* yellow (512-1023) */
for (i=1024; i<2048; ++i)
 mapcolor(i,255,0,255);   /* magenta (1024-2047)*/

color(BLACK);
clear();
swapbuffers();

} /* init_iris */
```

```
/*+------------------------------+
 |
 |   load_data.c
 |
 +------------------------------+              */

/* reads FHL elevation data from data file and inserts it into the global bottom_data array.
   Uses elevations as depths, so high elevation is a low depth */

#include "sonar.h"

load_bottom_data()
{
 int *fopen(), *fp;       /* file descriptor for the data file */
 short row, col;   /* loop indicies  */
 extern int bottom_data[BOTTOM_POINTS_WIDTH][BOTTOM_POINTS_HEIGHT];
 extern int max_depth, min_depth;

 /* read the data from the data file into the bottom_data array */
 fp = fopen("/work/hartley/sonar/fq5890-hr-e.da  ","r");
 for (col = 0; col < BOTTOM_POINTS_WIDTH; ++col)
 {
  for (row = 0; row < BOTTOM_POINTS_HEIGHT; ++row)
  {
   fscanf(fp, "%d",&bottom_data[row][col]);
   /* check if depth is a max or a min */
   if (bottom_data[row][col] > max_depth)
    max_depth = bottom_data[row][col];
   if (bottom_data[row][col] < min_depth)
    min_depth = bottom_data[row][col];
  } /* end inner for */
 } /* end outer for */

}/* load_bottom_data */
```

```
/*+-------------------------------+
 |
 |      make_arrows.c
 |
 +-------------------------------+              */

#include "sonar.h"
#include "math.h"
#include "gl.h"

make_arrows(x_coord, y_coord, depth, course, speed, set, drift,
 cog, sog, sweep_location, beam_length, tilt_angle, range_setting)

float x_coord, y_coord, depth, speed, sog, drift, beam_length;
int course, set, cog, tilt_angle, sweep_location, range_setting;
{

 /* varibles for location of arrow tips */
 static float course_arrow_x, course_arrow_y, current_arrow_x, current_arrow_y, cog_arrow_x,
  cog_arrow_y, l_course_x, l_course_y, r_course_x, r_course_y, l_current_x, l_current_y,
  r_current_x, r_current_y, l_cog_x, l_cog_y, r_cog_x, r_cog_y;

 /* convert y_coord to positive y */
 y_coord = -1 * y_coord;

 /* compute coords of arrow line segments */
 /* course arrow */
 course_arrow_x = x_coord + (ARROW_FACTOR * sin(course * DTOR) * speed);
 course_arrow_y = y_coord + (ARROW_FACTOR * cos(course * DTOR) * speed);
 l_course_x = course_arrow_x - (ARROW_WING_FACTOR * sin((course + 45) * DTOR) *
  speed);
 l_course_y = course_arrow_y - (ARROW_WING_FACTOR * cos((course + 45) * DTOR) *
  speed);
 r_course_x = course_arrow_x + (ARROW_WING_FACTOR * sin((course + 135) * DTOR) *
  speed);
 r_course_y = course_arrow_y + (ARROW_WING_FACTOR * cos((course + 135) * DTOR) *
  speed);

 /* current arrow */
 current_arrow_x = x_coord + (ARROW_FACTOR * sin(set * DTOR) * drift);
 current_arrow_y = y_coord + (ARROW_FACTOR * cos(set * DTOR) * drift);
 l_current_x = current_arrow_x - (ARROW_WING_FACTOR * sin((set + 45) * DTOR) * drift);
 l_current_y = current_arrow_y - (ARROW_WING_FACTOR * cos((set + 45) * DTOR) * drift);
 r_current_x = current_arrow_x + (ARROW_WING_FACTOR * sin((set + 135) * DTOR) * drift);
 r_current_y = current_arrow_y + (ARROW_WING_FACTOR * cos((set + 135) * DTOR) * drift);

 /* cog arrow */
 cog_arrow_x = course_arrow_x + (current_arrow_x - x_coord);
 cog_arrow_y = course_arrow_y + (current_arrow_y - y_coord);
 l_cog_x = cog_arrow_x - (ARROW_WING_FACTOR * sin((cog + 45) * DTOR) * sog);
 l_cog_y = cog_arrow_y - (ARROW_WING_FACTOR * cos((cog + 45) * DTOR) * sog);
 r_cog_x = cog_arrow_x + (ARROW_WING_FACTOR * sin((cog + 135) * DTOR) * sog);
 r_cog_y = cog_arrow_y + (ARROW_WING_FACTOR * cos((cog + 135) * DTOR) * sog);
```

```
viewport(647,1023,391,767);
ortho2(0.0,1000.0,0.0,1000.0);

/* delete old arrows */
color(BLACK);
clear();

linewidth(2);

/* make a red circle that shows the max beam range */
/* course arrow */
color(A_GREEN);
move2(x_coord,y_coord);
draw2(course_arrow_x, course_arrow_y);
draw2(l_course_x, l_course_y);
move2(course_arrow_x, course_arrow_y);
draw2(r_course_x, r_course_y);

/* current arrow */
color(A_YELLOW);
move2(x_coord, y_coord);
draw2(current_arrow_x, current_arrow_y);
draw2(l_current_x, l_current_y);
move2(current_arrow_x, current_arrow_y);
draw2(r_current_x, r_current_y);

/* cog arrow */
color(A_RED);
move2(x_coord, y_coord);
draw2(cog_arrow_x, cog_arrow_y);
draw2(l_cog_x, l_cog_y);
move2(cog_arrow_x, cog_arrow_y);
draw2(r_cog_x, r_cog_y);

/* put a red circle over the position */
color(A_RED);
circf(x_coord, y_coord,12);

/* make a red circle that shows the max beam range */
circ(x_coord, y_coord,(float)range_setting);

/* sonar sweep line */
color(A_MAGENTA);
pushmatrix();
translate(x_coord, y_coord);
rotate((sweep_location + course + 4) * -10, 'z');
move2(0.0,0.0);
draw2(beam_length * cos(tilt_angle * DTOR), 0.0);
popmatrix();

/* make a red circle that shows the max beam range */
circ(x_coord, y_coord,(float)range_setting);
```

```
} /* end make_arrows */
```

```
/*+-------------------------------+
 |
 |     make_chart.c
 |
 +-------------------------------+              */

/* make_chart.c - this procedure is called by main_sonar to build an object containing a chart map.
   The map is used for the full screen display when changing the location of the AUV, and in the
   upper right corner of the screen when sonar is operational.    */

#include "gl.h"
#include "sonar.h"
#include "math.h"

make_chart(chart)
 Object *chart;

{
 short i, j;
 float length, depth_float;
 int depth;
 /* vars used for color conversion from depth */
 Colorindex colour, lastcolor;
 /* terrain elevations */
 extern int bottom_data[BOTTOM_POINTS_WIDTH][BOTTOM_POINTS_HEIGHT];
 extern int max_depth, min_depth;

 *chart = genobj();  /* create the chart object */
 makeobj(*chart);
  viewport(0,647,120,767);
  linewidth(9); /* for large chart (5 for small chart)*/
  ortho2(0.0,80.0,0.0,80.0);    /* use array index space */

  color(BLACK);
  clear();

  lastcolor = BLACK;

  for (i=0; i < 80; ++i)
  {      /* draw column i */
   move2(i + 0.5,0.0); /* start at bottom of column  */
   length = 0.0;    /* # adjacent points of the same color */
   for (j=0; j< 80; ++j)
   {
    depth = bottom_data[j][i];
    depth_float = (float)depth;

    /* convert depth to a color */
    colour = (Colorindex)(MAX_COLOR_NUMBER - (((depth_float-min_depth)/
     (max_depth-min_depth)) * NUMBER_OF_COLORS));

    /* min color number is 16 */
    if (colour < 16) colour = 16;
```

```
 /* max color number is 63 */
 if (colour > 63) colour = 63;

 if (colour == lastcolor)
  length++;  /* don't draw yet */
 else
 {              /* draw now that color has changed */
  color(lastcolor);
  rdr2(0.0,length);
  lastcolor = colour;  /* reset for new draw */
  length = 1;
 }
 } /* end for j */

 color(colour);  /* draw last (top) line */
 rdr2(0.0,length);

 } /* end for i */

 linewidth(2);
 color(BLACK);              /* draw chart border */

 rect(0.0,0.0,80.0,80.0);

closeobj();

} /* end make_chart */
```

```
/*+-------------------------------+
 |
 |     make_depth.c
 |
 +-------------------------------+              */

/* make_depth_bar - this procedure is called by get_auv_settings to build the depth
   selection bar in the lower right hand corner of the screen */

#include "gl.h"
#include "sonar.h"

make_depth_bar(x_coord, y_coord, depth)
 float x_coord, y_coord, depth;

{
 int water_depth;
 char str[40];
 extern int bottom_data[BOTTOM_POINTS_WIDTH][BOTTOM_POINTS_HEIGHT];

 /* convert depth to ft */
 depth = depth * FEET_PER_M;

 /* calc water depth at current position */
 water_depth = bottom_data[(int)((-1 * y_coord)/RESOLUTION_IN_METERS)]
  [(int)(x_coord/RESOLUTION_IN_METERS)];

 /* set up box */
 viewport(647,1023,0,120);
 ortho2(0.0,100.0,0.0,33.0);
 color(CYAN);
 clear();

 color(BLACK);
 cmov2(39.0,29.0);
 charstr("0");
 cmov2(50.0,12.0);
 charstr("AUV DEPTH");
 cmov2(67.0,18.0);
 charstr("ft");

 /* print out auv depth */
 sprintf(str, "%5.1f", depth);
 cmov2(53.0,18.0);
 charstr(str);

 /* print out water depth on depth bar */
 sprintf(str, "%5d", water_depth);
 cmov2(36.0,2.0);
 charstr(str);

 /* draw the depth bar */
 color(WHITE);
 rectf(27.0,3.0,33.0,30.0);
```

```
color(BLUE);
linewidth(22);
if (depth != 0)
{
 move2(30.0,30.0);
 draw2(30.0, 30.0 - ((27.0 * depth)/water_depth));
}

/* border of depth bar */
linewidth(2);
color(BLACK);
rect(27.0,3.0,33.0,30.0);

} /* end make_depth_bar */
```

```
/*+-------------------------------+
 |
 |      make_dials.c
 |
 +-------------------------------+              */

/* make_dials - this procedure is called by main_sonar to build the dials on the sonar
   control panel */

#include "gl.h"
#include "sonar.h"

make_dials(dials, volume_tag, power_tag, sector_tag, range_tag)
Object *dials;
Tag *volume_tag, *power_tag, *sector_tag, *range_tag;
{
 *dials = genobj();
 makeobj(*dials);

 color(WHITE);

 /* make power dial */
 pushmatrix();
 translate(8.0,6.0);
 *power_tag = gentag();
 maketag(*power_tag);

 rotate(0,'z');

 translate(-8.0,-6.0);
 arcf(8.0,6.0,DIAL_RADIUS,2250,2350);
 popmatrix();

 /* make volume dial */
 pushmatrix();
 translate(41.6,6.0);

 *volume_tag = gentag();
 maketag(*volume_tag);
 rotate(0,'z');

 translate(-41.6,-6.0);
 arcf(41.6,6.0,DIAL_RADIUS,1350,1450);
 popmatrix();

 /* make near_gain dial */
 pushmatrix();
 translate(52.0,6.0);

 rotate(0,'z');

 translate(-52.0,-6.0);
 arcf(52.0,6.0,DIAL_RADIUS,2200,2300);
 popmatrix();
```

```
/* make far_gain dial */
pushmatrix();
translate(62.0,6.0);

rotate(0,'z');

translate(-62.0,-6.0);
arcf(62.0,6.0,DIAL_RADIUS,2200,2300);
popmatrix();

/* make sector dial */
pushmatrix();
translate(73.2,6.0);

*sector_tag = gentag();
maketag(*sector_tag);
rotate(0,'z');

translate(-73.2,-6.0);
arcf(73.2,6.0,DIAL_RADIUS,3100,3200);
popmatrix();

/* make range dial */
pushmatrix();
translate(88.4,6.0);

*range_tag = gentag();
maketag(*range_tag);
rotate(0,'z');

translate(-88.4,-6.0);
arcf(88.4,6.0,DIAL_RADIUS,3100,3200);
popmatrix();

closeobj();

} /* make_dials */
```

```
/*+-------------------------------+
 |
 |     make_inst.c
 |
 +-------------------------------+              */

/* make_instructions - this procedure is called by get_auv_settings to build the instruction
   billboard in the upper right hand corner of the screen */

#include "gl.h"
#include "sonar.h"

make_instructions(aground_flag)
 short aground_flag;

{
 short i; /* loop control */

 /* make exit sign at bottom */
 viewport (0,647,0,120);
 ortho2(0.0,100.0,0.0,18.0);

 color(BLUE);
 clear();

 color(WHITE);

 if (aground_flag)
 {

  cmov2(7.0,10.0);
  charstr("WHEN ALL AUV ATTRIBUTES HAVE DESIRED SETTINGS");
  cmov2(14.0,6.0);
  charstr("PRESS LEFT MOUSE BUTTON TO EXIT");
  /* draw aground flag */
  color(RED);
  rectf(82.0, 0.0,100.0,18.0);
  color(BLACK);
  cmov2(85.0,8.0);
  charstr("AGROUND!!");
 }
 else /* not aground */
 {
  cmov2(16.0,10.0);
  charstr("WHEN ALL AUV ATTRIBUTES HAVE DESIRED SETTINGS");
  cmov2(23.0,6.0);
  charstr("PRESS LEFT MOUSE BUTTON TO EXIT");
 }

 /* dials/mouse sign in upper right hand corner */
 viewport (647,1023,391,767);
 ortho2(0.0,100.0,0.0,100.0);

 color(MAGENTA);
```

```
clear();

color(WHITE);
cmov2(12.0,90.0);
charstr("SET AUV POSITION, COURSE & SPEED");
cmov2(19.0,82.0);
charstr("AND CURRENT SET & DRIFT");

/* dial box */
color(GREY);
rectf(16.0,16.0,48.0,76.0);
/* mouse box */
rectf(60.0,40.0,84.0,72.0);
/* dial box border, circles */
linewidth(2);
color(BLACK);
rect(16.0,16.0,48.0,76.0);
for (i=0; i< 4; ++i)
{
 circf(24.0, 23.0 + (i * 15), 4.0);
 circf(40.0, 23.0 + (i * 15), 4.0);
}
cmov2(18.0, 44.0);
charstr("COURSE");
cmov2(37.5, 44.0);
charstr("SET");
cmov2(20.0, 59.0);
charstr("DIVE");

/* mouse cord/ buttons */
/* cord */
rectf(71.0, 72.0,72.0, 76.0);
/* buttons */
for (i=0; i< 3; ++i)
{
 rectf(62.0 + (8 * i), 44.0,66.0 + (8 * i), 58.0);
}
/* mouse border */
rect(60.0,40.0,84.0,72.0);

/* writing at bottom of screen */
color(WHITE);
cmov2(23.0,7.0);
charstr("'DIALS'");

cmov2(60.0,31.0);
charstr("EXIT");

cmov2(70.0,27.0);
charstr("SET SPEED,");
cmov2(70.0,22.0);
charstr("DRIFT AND");
cmov2(62.0,16.0);
charstr("AUV LOCATION");
```

```
cmov2(65.0,6.5);
charstr("'MOUSE'");

/* arrows */
/* exit arrow */
move2(64.0, 36.0);
draw2(64.0, 42.0);
draw2(65.0, 40.5);
move2(64.0, 42.0);
draw2(63.0, 40.5);

/* set arrow */
move2(82.0, 32.0);
draw2(72.0, 42.0);
draw2(72.0, 40.5);
move2(72.0, 42.0);
draw2(73.5, 42.0);

} /* end make_instructions */
```

```
/*+-------------------------------+
 |
 |    make_scan_mask.c
 |
 +-------------------------------+              */

/* make_scan_mask - this procedure is called by main_sonar to
  build the scan mask that lays over sonar picture */

#include "gl.h"
#include "sonar.h"

make_scan_mask(scan_mask, scan_mask_tag)
Object *scan_mask;
Tag *scan_mask_tag;
{

 *scan_mask= genobj();
 makeobj(*scan_mask);

  color(BLACK);
  *scan_mask_tag = gentag();
  maketag (*scan_mask_tag);
  arcf(0.0,0.0,SONAR_RADIUS,0.0,0.0);

 closeobj();

} /* make_scan_mask */
```

```
/*+-------------------------------+
 |
 |     make_over.c
 |
 +-------------------------------+              */

/* make_range_ring_overlay - this procedure is called by make_sonar_video to build the
   4 range rings that over lay the sonar screen.  */

#include "gl.h"
#include "sonar.h"
#include "math.h"

make_range_ring_overlay()
{
 int i; /* loop variable */

 pushattributes();

 color(WHITE);
 for (i = 0; i < 360; i = i + 7)  /* outer circle 50 dots */
  circf(sin(i*DTOR)*SONAR_RADIUS_MINUS_A_LITTLE,cos(i*DTOR)*
   SONAR_RADIUS_MINUS_A_LITTLE,2);

 for (i=0; i<360; i = i + 9)  /* next to outer circle 40 dots*/
  circf(sin(i*DTOR)*THREE_QTR_SONAR_RADIUS,cos(i*DTOR)*
   THREE_QTR_SONAR_RADIUS,2);

 for (i=0; i<360; i = i + 14)  /* next to inner circle 25 dots */
  circf(sin(i*DTOR)*HALF_SONAR_RADIUS,cos(i*DTOR)*HALF_SONAR_RADIUS,2);

 for (i=0; i<360; i = i + 22) /* inner circle 16 dots */
  circf(sin(i*DTOR)*ONE_QTR_SONAR_RADIUS,cos(i*DTOR)*
   ONE_QTR_SONAR_RADIUS,2);

 popattributes();

} /* make_ring_ring_overlay */
```

```
/*+-------------------------------+
 |
 |     make_panel.c
 |
 +-------------------------------+              */

/* make_sonar_panel - this procedure is called by main_sonar to
  build the sonar cntl_panel with switches and dials. */

#include "gl.h"
#include "sonar.h"

make_sonar_panel(hoist_down, tilt_toggle_up, scan_toggle_up, range_toggle_up,
 volume_dial_location, power_dial_location, sector_dial_location, range_dial_location, dials)

 Object dials;
 short tilt_toggle_up, scan_toggle_up, range_toggle_up, hoist_down;
 int volume_dial_location, power_dial_location, sector_dial_location, range_dial_location;

{
 Coord n1[3][2], n2[4][2], n3[4][2], n4[4][2], n5[4][2], n6[4][2], n7[4][2], n8[4][2];
 Coord f1[3][2], f2[4][2], f3[4][2], f4[4][2], f5[4][2], f6[4][2], f7[4][2], f8[4][2];



 viewport(0,1073,0,120);
 ortho2(0.0,100.0,0.0,14.0);

 /* create the sonar panel backplate */
 color(GREY);
 rectf(0.0,1.8,100.0,11.8);
 color(BLACK);
 rectf(0.0,0.0,100.0,1.8);
 rectf(0.0,11.8,100.0,14.0);

 /* bottom line */
 cmov2(0.5,0.5);
 color(WHITE);
 charstr("WESMAR");
 cmov2(20.5,0.5);
 charstr("OMNICOLOR");
 cmov2(30.0,0.5);
 charstr("SCANNING SONAR SS265");
 cmov2(84.0,0.5);
 charstr("MADE IN USA");

 /* color bar */
 color(YELLOW);
 rectf(10.0,0.2,12.6,1.6);
 color(ORANGE);
 rectf(12.0,0.2,14.0,1.6);
 color(RED);
 rectf(14.0,0.2,16.0,1.6);
 color(GREEN);
```

```
rectf(16.0,0.2,18.0,1.6);
color(BLUE);
rectf(18.0,0.2,20.0,1.6);

/* dial headings */
color(WHITE);
cmov2(4.0,12.4);
charstr("VOLUME");
cmov2(14.0,12.4);
charstr("TILT");
cmov2(19.5,12.4);
charstr("SCAN POSIT.");
cmov2(29.0,12.4);
charstr("RANGE RING");
cmov2(39.5,12.4);
charstr("POWER");
cmov2(48.5,12.4);
charstr("NEAR GAIN");
cmov2(59.0,12.4);
charstr("FAR GAIN");
cmov2(71.0,12.4);
charstr("SECTOR");
cmov2(87.0,12.4);
charstr("RANGE");

/* hoist heading and indication circle */
cmov2(0.5,2.0);
charstr("HOIST");
if (!hoist_down)
 color(BLACK);
else
 color(RED);
circf(6.0,2.5,0.5);

/* tilt toggle */
color(WHITE);
cmov2(14.0,2.0);
charstr("DOWN");
cmov2(15.0,9.5);
charstr("UP");
linewidth(2);
color(BLACK);
rect(15.0,5.0,17.0,7.0);

/* scan heading toggle */
linewidth(2);
color(WHITE);
arc(24.0,6.0,3.7,600,1200);
arc(24.0,6.0,3.7,2400,3000);
move2(25.85, 9.2);
draw2(25.85,9.8);
move2(25.85, 9.2);
draw2(25.25,9.2);
move2(25.85, 2.8);
```

```
draw2(25.85,2.2);
move2(25.85, 2.8);
draw2(25.25,2.8);
color(BLACK);
rect(23.0,5.0,25.0,7.0);

/* range ring toggle */
color(WHITE);
cmov2(31.5,9.5);
charstr("IN");
cmov2(30.5,2.0);
charstr("OUT");
color(BLACK);
rect(31.0,5.0,33.0,7.0);

/* power dial heading */
color(WHITE);
cmov2(37.0,9.5);
charstr("OFF");
cmov2(44.0,9.5);
charstr("ON");

/* gain dial marks */
n1[0][0] = NEAR_CENTER_X - 3.4;
n1[0][1] = NEAR_CENTER_Y - 2.0;
n1[1][0] = NEAR_CENTER_X - 3.5;
n1[1][1] = NEAR_CENTER_Y - 0.3;
n1[2][0] = NEAR_CENTER_X - 3.3;
n1[2][1] = NEAR_CENTER_Y - 0.3;

n2[0][0] = NEAR_CENTER_X - 3.3;
n2[0][1] = NEAR_CENTER_Y - 0.0;
n2[1][0] = NEAR_CENTER_X - 3.5;
n2[1][1] = NEAR_CENTER_Y - 0.0;
n2[2][0] = NEAR_CENTER_X - 3.3;
n2[2][1] = NEAR_CENTER_Y + 1.7;
n2[3][0] = NEAR_CENTER_X - 2.9;
n2[3][1] = NEAR_CENTER_Y + 1.5;

n3[0][0] = NEAR_CENTER_X - 3.3;
n3[0][1] = NEAR_CENTER_Y + 1.9;
n3[1][0] = NEAR_CENTER_X - 2.9;
n3[1][1] = NEAR_CENTER_Y + 1.7;
n3[2][0] = NEAR_CENTER_X - 1.7;
n3[2][1] = NEAR_CENTER_Y + 3.0;
n3[3][0] = NEAR_CENTER_X - 2.0;
n3[3][1] = NEAR_CENTER_Y + 3.5;

n4[0][0] = NEAR_CENTER_X - 1.5;
n4[0][1] = NEAR_CENTER_Y + 3.1;
n4[1][0] = NEAR_CENTER_X - 1.8;
n4[1][1] = NEAR_CENTER_Y + 3.6;
n4[2][0] = NEAR_CENTER_X ;
n4[2][1] = NEAR_CENTER_Y + 4.1;
```

```
n4[3][0] = NEAR_CENTER_X ;
n4[3][1] = NEAR_CENTER_Y + 3.4;

n5[0][0] = NEAR_CENTER_X + 0.2;
n5[0][1] = NEAR_CENTER_Y + 3.4;
n5[1][0] = NEAR_CENTER_X + 0.2;
n5[1][1] = NEAR_CENTER_Y + 4.1;
n5[2][0] = NEAR_CENTER_X + 2.0;
n5[2][1] = NEAR_CENTER_Y + 3.8;
n5[3][0] = NEAR_CENTER_X + 1.5;
n5[3][1] = NEAR_CENTER_Y + 3.0;

n6[0][0] = NEAR_CENTER_X + 1.8;
n6[0][1] = NEAR_CENTER_Y + 2.9;
n6[1][0] = NEAR_CENTER_X + 2.1;
n6[1][1] = NEAR_CENTER_Y + 3.7;
n6[2][0] = NEAR_CENTER_X + 3.5;
n6[2][1] = NEAR_CENTER_Y + 2.6;
n6[3][0] = NEAR_CENTER_X + 2.8;
n6[3][1] = NEAR_CENTER_Y + 2.1;

n7[0][0] = NEAR_CENTER_X + 2.9;
n7[0][1] = NEAR_CENTER_Y + 2.0;
n7[1][0] = NEAR_CENTER_X + 3.6;
n7[1][1] = NEAR_CENTER_Y + 2.3;
n7[2][0] = NEAR_CENTER_X + 4.4;
n7[2][1] = NEAR_CENTER_Y ;
n7[3][0] = NEAR_CENTER_X + 3.2;
n7[3][1] = NEAR_CENTER_Y ;

n8[0][0] = NEAR_CENTER_X + 3.2;
n8[0][1] = NEAR_CENTER_Y - 0.3;
n8[1][0] = NEAR_CENTER_X + 4.4;
n8[1][1] = NEAR_CENTER_Y - 0.3;
n8[2][0] = NEAR_CENTER_X + 4.6;
n8[2][1] = NEAR_CENTER_Y - 2.0;
n8[3][0] = NEAR_CENTER_X + 2.9;
n8[3][1] = NEAR_CENTER_Y - 2.0;

f1[0][0] = FAR_CENTER_X - 3.4;
f1[0][1] = FAR_CENTER_Y - 2.0;
f1[1][0] = FAR_CENTER_X - 3.5;
f1[1][1] = FAR_CENTER_Y - 0.3;
f1[2][0] = FAR_CENTER_X - 3.3;
f1[2][1] = FAR_CENTER_Y - 0.3;

f2[0][0] = FAR_CENTER_X - 3.3;
f2[0][1] = FAR_CENTER_Y - 0.0;
f2[1][0] = FAR_CENTER_X - 3.5;
f2[1][1] = FAR_CENTER_Y - 0.0;
f2[2][0] = FAR_CENTER_X - 3.3;
f2[2][1] = FAR_CENTER_Y + 1.7;
f2[3][0] = FAR_CENTER_X - 2.9;
f2[3][1] = FAR_CENTER_Y + 1.5;
```

```
f3[0][0] = FAR_CENTER_X - 3.3;
f3[0][1] = FAR_CENTER_Y + 1.9;
f3[1][0] = FAR_CENTER_X - 2.9;
f3[1][1] = FAR_CENTER_Y + 1.7;
f3[2][0] = FAR_CENTER_X - 1.7;
f3[2][1] = FAR_CENTER_Y + 3.0;
f3[3][0] = FAR_CENTER_X - 2.0;
f3[3][1] = FAR_CENTER_Y + 3.5;

f4[0][0] = FAR_CENTER_X - 1.5;
f4[0][1] = FAR_CENTER_Y + 3.1;
f4[1][0] = FAR_CENTER_X - 1.8;
f4[1][1] = FAR_CENTER_Y + 3.6;
f4[2][0] = FAR_CENTER_X ;
f4[2][1] = FAR_CENTER_Y + 4.1;
f4[3][0] = FAR_CENTER_X ;
f4[3][1] = FAR_CENTER_Y + 3.4;

f5[0][0] = FAR_CENTER_X + 0.2;
f5[0][1] = FAR_CENTER_Y + 3.4;
f5[1][0] = FAR_CENTER_X + 0.2;
f5[1][1] = FAR_CENTER_Y + 4.1;
f5[2][0] = FAR_CENTER_X + 2.0;
f5[2][1] = FAR_CENTER_Y + 3.8;
f5[3][0] = FAR_CENTER_X + 1.5;
f5[3][1] = FAR_CENTER_Y + 3.0;

f6[0][0] = FAR_CENTER_X + 1.8;
f6[0][1] = FAR_CENTER_Y + 2.9;
f6[1][0] = FAR_CENTER_X + 2.1;
f6[1][1] = FAR_CENTER_Y + 3.7;
f6[2][0] = FAR_CENTER_X + 3.5;
f6[2][1] = FAR_CENTER_Y + 2.6;
f6[3][0] = FAR_CENTER_X + 2.8;
f6[3][1] = FAR_CENTER_Y + 2.1;

f7[0][0] = FAR_CENTER_X + 2.9;
f7[0][1] = FAR_CENTER_Y + 2.0;
f7[1][0] = FAR_CENTER_X + 3.6;
f7[1][1] = FAR_CENTER_Y + 2.3;
f7[2][0] = FAR_CENTER_X + 4.4;
f7[2][1] = FAR_CENTER_Y ;
f7[3][0] = FAR_CENTER_X + 3.2;
f7[3][1] = FAR_CENTER_Y ;

f8[0][0] = FAR_CENTER_X + 3.2;
f8[0][1] = FAR_CENTER_Y - 0.3;
f8[1][0] = FAR_CENTER_X + 4.4;
f8[1][1] = FAR_CENTER_Y - 0.3;
f8[2][0] = FAR_CENTER_X + 4.6;
f8[2][1] = FAR_CENTER_Y - 2.0;
f8[3][0] = FAR_CENTER_X + 2.9;
f8[3][1] = FAR_CENTER_Y - 2.0;
```

```
/* near gain dial marks */
polf2(3,n1);
polf2(4,n2);
polf2(4,n3);
polf2(4,n4);
polf2(4,n5);
polf2(4,n6);
polf2(4,n7);
polf2(4,n8);

/* far gain dial marks */
polf2(3,f1);
polf2(4,f2);
polf2(4,f3);
polf2(4,f4);
polf2(4,f5);
polf2(4,f6);
polf2(4,f7);
polf2(4,f8);

/* sector dial indicating arcs */
arcf(69.4,3.0,1.2,825,975);
arcf(69.4,5.1,1.2,750,1050);
arcf(69.6,7.9,1.2,675,1125);
arcf(71.8,9.2,1.2,450,1350);
arcf(74.3,9.2,1.2,225,1575);
arcf(77.0,8.0,0.8,0,1800);
arcf(77.3,6.1,0.7,3150,2250);
circf(77.1,3.8,0.7);

/* range dial marks */
circf(85.6,3.2,0.1);
cmov2(82.8,2.0);
charstr("15");
circf(84.4,5.0,0.1);
cmov2(81.8,4.0);
charstr("30");
circf(84.1,7.0,0.1);
cmov2(81.4,6.3);
charstr("60");
circf(85.0,9.0,0.1);
cmov2(81.4,8.3);
charstr("100");
circf(87.1,10.1,0.1);
cmov2(84.3,10.0);
charstr("150");
circf(89.1,10.1,0.1);
cmov2(90.1,10.0);
charstr("200");
circf(91.2,9.0,0.1);
cmov2(92.8,8.3);
charstr("300");
circf(92.1,7.0,0.1);
```

```
cmov2(93.1,6.3);
charstr("400");
circf(91.8,5.0,0.1);
cmov2(92.8,4.0);
charstr("600");
circf(90.6,3.2,0.1);
cmov2(92.3,2.0);
charstr("800");

/* panel borders */
color(WHITE);
linewidth(2);
rect(0.0,0.0,100.0,14.0);
rect(0.0,1.8,100.0,11.7);

/* build toggles */
if (tilt_toggle_up)
 edit_toggle(TILT,UP);
else
 edit_toggle(TILT,DOWN);

if (scan_toggle_up)
 edit_toggle(SCAN,UP);
else
 edit_toggle(SCAN,DOWN);

if (range_toggle_up)
 edit_toggle(RANGE_RING,UP);
else
 edit_toggle(RANGE_RING,DOWN);

/* build dial circles */
color(BLACK);
circf(8.0,6.0,DIAL_RADIUS); /* vol */
circf(41.6,6.0,DIAL_RADIUS);/* power */
circf(52.0,6.0,DIAL_RADIUS);/* near gain */
circf(62.0,6.0,DIAL_RADIUS);/* far gain*/
circf(73.2,6.0,DIAL_RADIUS);/* sector */
circf(88.4,6.0,DIAL_RADIUS);/* range */

/* call the white filled arcs over the black dial circles
 to indicate dial locations */
callobj(dials);

} /* make_sonar_panel */
```

```
/*+-------------------------------+
 |
 |     make_plus.c
 |
 +-------------------------------+              */

/* make_position_plus - this procedure is called by get_auv_settings to build the position_plus
  on the large chart on the left of the screen (the plus marks the location of the auv)*/

#include "gl.h"
#include "sonar.h"

make_position_plus(x_coord, y_coord)
 float x_coord, y_coord;

{
 /* convert -y to pos y */
 y_coord = -1 * y_coord;

 viewport (0,647,120,767);
 ortho2(0.0,1000.0,0.0,1000.0);

 /* clear previous pluses */
 color(BLACK);
 clear();

 linewidth(2);
 color(A_RED);

 /* draw the plus over the position */
 move2(x_coord, y_coord);
 draw2(x_coord + PLUS_FACTOR, y_coord);
 move2(x_coord, y_coord);
 draw2(x_coord - PLUS_FACTOR, y_coord);
 move2(x_coord, y_coord);
 draw2(x_coord, y_coord + PLUS_FACTOR);
 move2(x_coord, y_coord);
 draw2(x_coord, y_coord - PLUS_FACTOR);

 /* decide where to put the "current position" label */
 if ((x_coord < 500) && (y_coord <= 500))
  cmov2(x_coord + PLUS_FACTOR, y_coord + PLUS_FACTOR);
 else if ((x_coord < 500) && (y_coord > 500))
  cmov2(x_coord + PLUS_FACTOR, y_coord - PLUS_FACTOR);
 else if ((x_coord >= 500) && (y_coord <= 500))
  cmov2(x_coord - (7 * PLUS_FACTOR), y_coord + PLUS_FACTOR);
 else
  cmov2(x_coord - (7 * PLUS_FACTOR), y_coord - PLUS_FACTOR);

 charstr("CURRENT POSITION");

} /* end make_position_plus */
```

```
/*+-------------------------------+
 |
 |     make_read.c
 |
 +-------------------------------+              */

/* make_readout - this procedure is called by main_sonar and get_auv_settings to build the auv
   instrument readout on the right hand corner of the screen */

#include "gl.h"
#include "sonar.h"

make_readout(scan_mode, x_coord, y_coord, depth, course, speed, dive_angle, selected_course,
 selected_speed, selected_dive_angle, set, drift, cog, sog, tilt_inc)

 float x_coord, y_coord, depth, speed, drift, sog, selected_speed;
 int course, set, dive_angle, cog, selected_course, selected_dive_angle, tilt_inc;
 short scan_mode;

{
 char str[40];
 short i; /* loop control */

 viewport (647,1023,120,391);
 ortho2(0.0,100.0,0.0,74.0);

 color(BLACK);
 clear();

 /* course and speed */
 color(GREEN);
 rectf(0.0,37.5,66.0,74.0);
 color(WHITE);
 circf(13.0,62.0,9.0);
 /* speed bar*/
 rectf(29.75,53.0,36.25,71.0);
 /* course box*/
 rectf(8.0,45.0,18.0,50.0);
 /* speed box*/
 rectf(28.0,45.0,38.0,50.0);
 /* dive angle arc */
 arcf(44.0,62.0,9.0,2700,900);
 /* dive box */
 rectf(48.0,45.0,58.0,50.0);

 /* indicator titles */
 color(BLACK);
 cmov2(6.0,40.0);
 charstr("COURSE");
 cmov2(26.0,40.0);
 charstr("SPEED");
 cmov2(42.0,40.0);
 charstr("DIVE ANGLE");
 /* labels on speed bar */
```

```
/* high mark */
/*sprintf(str, "%5.1f", MAX_SPEED);*/
sprintf(str, "%2d", 12);
cmov2(24.0,70.0);
charstr(str);
/* low mark */
/*sprintf(str, "%5.1f", MIN_SPEED);*/
sprintf(str, "%d", -4);
cmov2(24.0,52.0);
charstr(str);
/* zero mark */
linewidth(1);
move2(29.0, ZERO_Y);
draw2(36.25, ZERO_Y);
cmov2(25.0, ZERO_Y - 1.0);
charstr("0");

/* dive angle labels */
cmov2(55.0, 68.0);
charstr("UP");
cmov2(56.0, 61.0);
charstr("0");
cmov2(55.0, 53.0);
charstr("DOWN");

linewidth(2);
/* outline course, speed and dive angle arcs and boxes */
/* course circle */
circ(13.0,62.0,9.0);
/* course box */
rect(8.0,45.0,18.0,50.0);
/* speed box */
rect(28.0,45.0,38.0,50.0);
/* speed bar */
rect(29.75,53.0,36.25,71.0);
/* dive angle arc */
move2(44.0,71.0);
draw2(44.0, 53.0);
arc(44.0, 62.0,9.0,2700,900);
rect(48.0,45.0,58.0,50.0);

/* put the numbers in the boxes */
/* course */
sprintf(str, "%3d", course);
cmov2(9.0,46.0);
charstr(str);
/* degree circle */
circ(16.5,48.5,0.50);

/* speed */
sprintf(str, "%3.1f", speed);
cmov2(28.7,46.0);
charstr(str);
```

```
/* dive angle */
sprintf(str, "%3d", dive_angle);
cmov2(49.0,46.0);
charstr(str);
/* degree circle */
circ(56.5,48.5,0.50);

/* draw in speed bar */
linewidth(24);
if (selected_speed != 0)
{
 move2(33.0, ZERO_Y);
 draw2(33.0, ZERO_Y + (selected_speed * SPEED_INC));
}
linewidth(2);

/* draw in course indicator */
pushmatrix();
translate(13.0, 62.0);
rotate(selected_course * -10, 'z');
translate(-13.0,-62.0);
move2(13.0,62.0);
draw2(13.0, 71.0);
popmatrix();

/* draw in dive_angle indicator */
pushmatrix();
translate(44.0, 62.0);
rotate(selected_dive_angle * -10, 'z');
translate(-44.0,-62.0);
move2(44.0,62.0);
draw2(53.0,62.0);
popmatrix();

/* now display current info */
color(YELLOW);
rectf(0.0,0.0,66.0,37.0);
color(WHITE);
circf(17.0,25.0,9.0);
/* drift bar */
rectf(46.0,16.0,53.0,34.0);
/* set box */
rectf(11.0,8.0,24.0,13.0);
/* drift box */
rectf(43.0,8.0,57.0,13.0);
color(BLACK);
/* outline boxes */
/* set box */
rect(11.0,8.0,24.0,13.0);
/* drift box */
rect(43.0,8.0,57.0,13.0);
/* set circle border*/
circ(17.0,25.0,9.0);
```

```
cmov2(13.0,3.0);
charstr("SET");
cmov2(43.0,3.0);
charstr("DRIFT");
sprintf(str, "%d", MAX_DRIFT);
cmov2(43.0,33.0);
charstr(str);
cmov2(43.0,15.0);
charstr("0");

/* set readout */
sprintf(str, "%3d", set);
cmov2(13.0,9.0);
charstr(str);
/* degree circle */
circ(22.5,11.5,0.50);

/* drift readout */
sprintf(str, "%3.1f", drift);
cmov2(46.0,9.0);
charstr(str);

/* drift bar */
linewidth(24);
if (drift != 0)
{
 move2(49.5,16.0);
 draw2(49.5,16.0 + (DRIFT_INC * drift));
}
linewidth(2);
rect(46.0,16.0,53.0,34.0);

/* set circle indicator */
pushmatrix();
translate(17.0,25.0);
rotate(set * -10, 'z');
translate(-17.0, -25.0);
move2(17.0,25.0);
draw2(17.0,34.0);
popmatrix();

/* draw the numbei readout now */
/* AUV position */
color(BLUE);
rectf(66.0,38.0,100.0,74.0);
/* AUV cog sog */
color(RED);
rectf(66.0,14.0,100.0,38.0);
/* scan mode and tilt_inc */
color(MAGENTA);
rectf(66.0,0.0,100.0,14.0);
color(BLACK);
for (i=0; i< 4; ++i)
```

```
{
 move2(66.0, (i*12) + 26.0);
 draw2(100.0, (i*12) + 26.0);
}
move2(66.0, 14.0);
draw2(100.0, 14.0);

/* frame the readout */
rect(66.0,0.0,100.0,74.0);
/* seperate the AUV display from the ocean CURRENT display */
move2(0.0,37.0);
draw2(66.0,37.0);

color(WHITE);
/* number boxes */
for (i=0; i< 5; ++i)
{
 rectf(78.0,20.0 + (i * 12), 90.0, 25.0 + (i * 12));
}

/* enter numbers */
color(BLACK);
sprintf(str, "%5.1f", x_coord);
cmov2(78.5,69.0);
charstr(str);
sprintf(str, "%5.1f", (-1 * y_coord));
cmov2(78.5,57.0);
charstr(str);
sprintf(str, "%5.1f", depth);
cmov2(78.5,45.0);
charstr(str);
sprintf(str, "%3.1f", sog);
cmov2(81.0,21.0);
charstr(str);
sprintf(str, "%3d", cog);
cmov2(79.0,33.0);
charstr(str);
circ(89.0,36.0,0.50);

/* now label boxes */
color(WHITE);
cmov2(76.0,64.0);
charstr("X-COORD");
cmov2(76.0,52.0);
charstr("Y-COORD");
cmov2(67.0,40.0);
charstr("AUV DEPTH (m)");
cmov2(80.0,28.0);
charstr("COG");
cmov2(74.0,16.0);
charstr("SOG (kts)");

color(BLACK);
cmov2(70.0,8.0);
```

```
  charstr("SCAN MODE:");
  cmov2(94.5,8.0);
  if (scan_mode == JUST_SCAN_NO_STORE)
   charstr("0");
  else if (scan_mode == ONE_SCAN_AND_STORE)
   charstr("1");
  else
   charstr("2");
 cmov2(70.0,3.0);
 charstr("TILT INC:");
 sprintf(str, "%2d", tilt_inc);
 cmov2(92.0,3.0);
 charstr(str);

} /* end make_readout */
```

```
/*+-------------------------------+
 |
 |    make_ring.c
 |
 +-------------------------------+              */

/* make_range_ring.c - this procedure is called by make_sonar_video to build the range ring that
   moves in and out on the sonar screen, as the range toggle is incremented or decremented */

#include "gl.h"
#include "sonar.h"
#include "math.h"

make_range_r.ng(range_ring_location)
int range_ring_location;
{
 short i; /* loop variable */
 short dot_increment;

 color(WHITE);
 if (range_ring_location < ONE_QTR_SONAR_RADIUS)
  dot_increment = 22;
 else if (range_ring_location < HALF_SONAR_RADIUS)
  dot_increment = 14;
 else if (range_ring_location < THREE_QTR_SONAR_RADIUS)
  dot_increment = 9;
 else dot_increment = 7;
 for (i=0; i<360; i = i + dot_increment)
 {
  circf(sin(i*DTOR)*range_ring_location,cos(i*DTOR)* range_ring_location,2);
 }

} /* make_range_ring  */
```

```
/*+-------------------------------+
 |
 |     make_rtn.c
 |
 +-------------------------------+              */

/*make_video_rtn - this procedure is called by main_sonar to build the sonar video return image*/

#include "gl.h"
#include "sonar.h"

make_video_rtn(video_arc_array, arc_tag_array)
Object video_arc_array[45]; /* array of 8 degree arc objects */
Tag arc_tag_array[][15]; /* array of 15 tags for each object */
{
 int i;   /* loop control */

 for (i=0; i<45; ++i)
 {

  video_arc_array[i]= genobj();
  makeobj(video_arc_array[i]);

   pushmatrix();

   rotate(-40,'z');
   rotate((i + 1) * -80, 'z');

   /* make one 8 degree arc and then make 45 8 degree arcs by rotating this one in 8 degree
      increments. Then rotate each arc another 4 degrees so center of arc 0 will lie at
      0 degrees and others will be centered on 8 degree increments */
   one_sweep(i, arc_tag_array);

   popmatrix();

  closeobj();

 } /* end for */

} /* make_video_rtn */

/* procedure one sweep creates one 8 degree sonar video arc */
one_sweep(video_arc_index, arc_tag_array)
 Tag arc_tag_array[][15];
 int video_arc_index;

 {
 Coord p[31][2];
 Coord a0[3][2], a1[4][2], a2[4][2], a3[4][2], a4[4][2], a5[4][2], a6[4][2], a7[4][2], a8[4][2],
  a9[4][2], a10[4][2], a11[4][2], a12[4][2], a13[4][2], a14[4][2];
 int i;


 /* build points along 0 degree line */
```

```
for (i=0; i<16; ++i)
{
 p[i][0] = i * 2.93333;
 p[i][1] = 0;
}

/* build points along 8 degree line */
for (i=16; i<31; ++i)
{
 p[i][0] = (i - 15) * 2.90479; /* cos-8 * 2.9333 */
 p[i][1] = (i - 15) * 0.408236; /* sin-8 * 2.9333 */
}

 /* closest polygon to origin */
 arc_tag_array[video_arc_index][0] = gentag();
 maketag (arc_tag_array[video_arc_index][0]);
 color(NO_CONTACT_COLOR);
 a0[0][0] = p[0][0];
 a0[0][1] = p[0][1];
 a0[1][0] = p[1][0];
 a0[1][1] = p[1][1];
 a0[2][0] = p[16][0];
 a0[2][1] = p[16][1];

 polf2(3,a0);

 arc_tag_array[video_arc_index][1] = gentag();
 maketag (arc_tag_array[video_arc_index][1]);
 color(NO_CONTACT_COLOR);
 a1[0][0] = p[16][0];
 a1[0][1] = p[16][1];
 a1[1][0] = p[1][0];
 a1[1][1] = p[1][1];
 a1[2][0] = p[2][0];
 a1[2][1] = p[2][1];
 a1[3][0] = p[17][0];
 a1[3][1] = p[17][1];

 polf2(4,a1);

 arc_tag_array[video_arc_index][2] = gentag();
 maketag (arc_tag_array[video_arc_index][2]);
 color(NO_CONTACT_COLOR);
 a2[0][0] = p[17][0];
 a2[0][1] = p[17][1];
 a2[1][0] = p[2][0];
 a2[1][1] = p[2][1];
 a2[2][0] = p[3][0];
 a2[2][1] = p[3][1];
 a2[3][0] = p[18][0];
 a2[3][1] = p[18][1];

 polf2(4,a2);
```

```
arc_tag_array[video_arc_index][3] = gentag();
maketag (arc_tag_array[video_arc_index][3]);
color(NO_CONTACT_COLOR);
a3[0][0] = p[3][0];
a3[0][1] = p[3][1];
a3[1][0] = p[4][0];
a3[1][1] = p[4][1];
a3[2][0] = p[19][0];
a3[2][1] = p[19][1];
a3[3][0] = p[18][0];
a3[3][1] = p[18][1];

polf2(4,a3);

arc_tag_array[video_arc_index][4] = gentag();
maketag (arc_tag_array[video_arc_index][4]);
color(NO_CONTACT_COLOR);
a4[0][0] = p[4][0];
a4[0][1] = p[4][1];
a4[1][0] = p[5][0];
a4[1][1] = p[5][1];
a4[2][0] = p[20][0];
a4[2][1] = p[20][1];
a4[3][0] = p[19][0];
a4[3][1] = p[19][1];

polf2(4,a4);

arc_tag_array[video_arc_index][5] = gentag();
maketag (arc_tag_array[video_arc_index][5]);
color(NO_CONTACT_COLOR);
a5[0][0] = p[5][0];
a5[0][1] = p[5][1];
a5[1][0] = p[6][0];
a5[1][1] = p[6][1];
a5[2][0] = p[21][0];
a5[2][1] = p[21][1];
a5[3][0] = p[20][0];
a5[3][1] = p[20][1];

polf2(4,a5);

arc_tag_array[video_arc_index][6] = gentag();
maketag (arc_tag_array[video_arc_index][6]);
color(NO_CONTACT_COLOR);
a6[0][0] = p[21][0];
a6[0][1] = p[21][1];
a6[1][0] = p[6][0];
a6[1][1] = p[6][1];
a6[2][0] = p[7][0];
a6[2][1] = p[7][1];
a6[3][0] = p[22][0];
a6[3][1] = p[22][1];
```

```
polf2(4,a6);

arc_tag_array[video_arc_index][7] = gentag();
maketag (arc_tag_array[video_arc_index][7]);
color(NO_CONTACT_COLOR);
a7[0][0] = p[7][0];
a7[0][1] = p[7][1];
a7[1][0] = p[8][0];
a7[1][1] = p[8][1];
a7[2][0] = p[23][0];
a7[2][1] = p[23][1];
a7[3][0] = p[22][0];
a7[3][1] = p[22][1];

polf2(4,a7);

arc_tag_array[video_arc_index][8] = gentag();
maketag (arc_tag_array[video_arc_index][8]);
color(NO_CONTACT_COLOR);
a8[0][0] = p[8][0];
a8[0][1] = p[8][1];
a8[1][0] = p[9][0];
a8[1][1] = p[9][1];
a8[2][0] = p[24][0];
a8[2][1] = p[24][1];
a8[3][0] = p[23][0];
a8[3][1] = p[23][1];

polf2(4,a8);

arc_tag_array[video_arc_index][9] = gentag();
maketag (arc_tag_array[video_arc_index][9]);
color(NO_CONTACT_COLOR);
a9[0][0] = p[9][0];
a9[0][1] = p[9][1];
a9[1][0] = p[10][0];
a9[1][1] = p[10][1];
a9[2][0] = p[25][0];
a9[2][1] = p[25][1];
a9[3][0] = p[24][0];
a9[3][1] = p[24][1];

polf2(4,a9);

arc_tag_array[video_arc_index][10] = gentag();
maketag (arc_tag_array[video_arc_index][10]);
color(NO_CONTACT_COLOR);
a10[0][0] = p[10][0];
a10[0][1] = p[10][1];
a10[1][0] = p[11][0];
a10[1][1] = p[11][1];
a10[2][0] = p[26][0];
a10[2][1] = p[26][1];
a10[3][0] = p[25][0];
```

```
a10[3][1] = p[25][1];

polf2(4,a10);

arc_tag_array[video_arc_index][11] = gentag();
maketag (arc_tag_array[video_arc_index][11]);
color(NO_CONTACT_COLOR);
a11[0][0] = p[11][0];
a11[0][1] = p[11][1];
a11[1][0] = p[12][0];
a11[1][1] = p[12][1];
a11[2][0] = p[27][0];
a11[2][1] = p[27][1];
a11[3][0] = p[26][0];
a11[3][1] = p[26][1];

polf2(4,a11);

arc_tag_array[video_arc_index][12] = gentag();
maketag (arc_tag_array[video_arc_index][12]);
color(NO_CONTACT_COLOR);
a12[0][0] = p[12][0];
a12[0][1] = p[12][1];
a12[1][0] = p[13][0];
a12[1][1] = p[13][1];
a12[2][0] = p[28][0];
a12[2][1] = p[28][1];
a12[3][0] = p[27][0];
a12[3][1] = p[27][1];

polf2(4,a12);

arc_tag_array[video_arc_index][13] = gentag();
maketag (arc_tag_array[video_arc_index][13]);
color(NO_CONTACT_COLOR);
a13[0][0] = p[13][0];
a13[0][1] = p[13][1];
a13[1][0] = p[14][0];
a13[1][1] = p[14][1];
a13[2][0] = p[29][0];
a13[2][1] = p[29][1];
a13[3][0] = p[28][0];
a13[3][1] = p[28][1];

polf2(4,a13);

/* polygon out at sonar radius */
arc_tag_array[video_arc_index][14] = gentag();
maketag (arc_tag_array[video_arc_index][14]);
color(NO_CONTACT_COLOR);
a14[0][0] = p[29][0];
a14[0][1] = p[29][1];
a14[1][0] = p[14][0];
a14[1][1] = p[14][1];
```

```
 a14[2][0] = p[15][0];
 a14[2][1] = p[15][1];
 a14[3][0] = p[30][0];
 a14[3][1] = p[30][1];

 polf2(4,a14);

} /* end one_sweep */
```

```
/*+-------------------------------+
 |
 |    make_scline.c
 |
 +-------------------------------+              */

/* make_scan_heading_line - this procedure is called by main_sonar
  to build the scan heading line(direction the sonar is pointed) */

#include "gl.h"
#include "sonar.h"

make_scan_heading_line(scan_heading_line, scan_heading_tag)
Object *scan_heading_line;
Tag *scan_heading_tag;
{
 *scan_heading_line= genobj();
 makeobj(*scan_heading_line);

  pushmatrix();
  color(WHITE);
  linewidth(2);

  *scan_heading_tag = gentag();
  maketag (*scan_heading_tag);
  rotate(0,'z');
  move2(0.0,0.0);
  draw2(SONAR_RADIUS, 0.0);
  popmatrix();

 closeobj();

} /* make_scan_heading_line */
```

```
/*+-------------------------------+
 |
 |    make_sweep.c
 |
 +-------------------------------+              */

/* make_sonar_sweep - this procedure is called by main_sonar to build the sonar sweep line*/

#include "gl.h"
#include "sonar.h"

make_sonar_sweep(sonar_sweep, sonar_sweep_tag)
Object *sonar_sweep;
Tag *sonar_sweep_tag;
{

 *sonar_sweep= genobj();

 makeobj(*sonar_sweep);

  pushmatrix();
  linewidth(2);
  color(WHITE);
  /* rotate 4 degrees clockwise to coorespond to same rotation of video arc objects */
  rotate(-40,'z');

  *sonar_sweep_tag = gentag();
  maketag (*sonar_sweep_tag);
  rotate(0,'z');
  move2(0.0,0.0);
  draw2(43.57, -6.12);
  popmatrix();

 closeobj();

} /* make_sonar_sweep */
```

```
/*+-------------------------------+
 |
 |     make_tilt.c
 |
 +-------------------------------+              */

/* make_tilt_angle_indicator - this procedure is called by main_sonar to build the tilt angle
   indicator in the upper left hand corner of the sonar video screen */

#include "gl.h"
#include "sonar.h"

make_tilt_angle_indicator(tilt_angle_indicator, tilt_angle_tag)
Object *tilt_angle_indicator;
Tag *tilt_angle_tag;
{

 *tilt_angle_indicator= genobj();
 makeobj(*tilt_angle_indicator);

  pushmatrix();
  color(WHITE);
  linewidth(3);
  translate(-46.0,36.0);

  *tilt_angle_tag = gentag();
  maketag (*tilt_angle_tag);
  rotate(0,'z');

  translate(46.0,-36.0);
  move2(-46.0,36.0);
  draw2(-34.0,36.0);
  popmatrix();

 closeobj();

} /* make_tilt_angle_indicator */
```

```
/*+-------------------------------+
 |
 |     make_video.c
 |
 +-------------------------------+              */

/* make_sonar_video - this procedure is called by main_sonar
to build the sonar video monitor with indicators. */

#include "gl.h"
#include "sonar.h"
#include "math.h"

make_sonar_video(sonar_sweep, range_setting, tilt_angle, range_ring_location, hoist_down,
 power_on, x_coord, y_coord, depth, tilt_angle_indicator, scan_mask, scan_heading_line,
 range_setting_change, sector_setting, video_arc_array, arc_tag_array, course, dive_angle,
 roll_angle, sweep_location, sweep_clockwise, beam_length, beam_inc, encountered_contact)

 Object tilt_angle_indicator, scan_mask, scan_heading_line, sonar_sweep, video_arc_array[45];

 Tag arc_tag_array[][15];

 int range_setting, sector_setting, tilt_angle, range_ring_location;
 short power_on, hoist_down, range_setting_change;
 float x_coord, y_coord, depth, *beam_length, *beam_inc;
 int course, dive_angle, roll_angle, *sweep_location;
 short sweep_clockwise; /* boolean for sweep direction */
 short *encountered_contact;

 {
 extern int bottom_data[BOTTOM_POINTS_WIDTH][BOTTOM_POINTS_HEIGHT];
 int depth_at_sweep_point;
 short hun_digit, tens_digit, ones_digit;
 int pos_tilt; /* negative tilt converted to positive number */
 int depth_value;
 int temp_range_ring_location; /* temp holder for converted rr loc. */
 short i; /* loop control */
 static short sonar_running = FALSE;

 /* varibles for Euler angles */
 float c4,c5,c6,c7,c8,s4,s5,s6,s7,s8;
 short boundary_flag; /* boolean for database boundary hit */
 int  depth_at_tip, arc_tag, arc_index;
 /* function to return db depth value at beam tip point */
 int bottom_depth_at_tip();
 float find_bottom_inc(), /* increase beam function */
  find_bottom_dec(), /* decrease beam function */
  tip_depth(); /* depth of beam tip function */

 /* convert dive_angle for use in D-H matricies */
 /* dive_angle down is positive from readout, but negative for D-H */
 dive_angle = -1 * dive_angle;

 viewport(0,647,120,767);  /*  upper left hand corner of screen */
```

```
ortho2(-50.0,50.0,-50.0,50.0);
color(BLACK);
clear();

/* video border */
color(WHITE);
linewidth(2);
rect(-49.0,-50.0,50.0,50.0);

if (!sonar_running && power_on && hoist_down)
{
 /* whenever the sonar is turned on, reset sweep_location to 0 */
 *sweep_location = 0;
 range_setting_change = TRUE;
}

if (!power_on || !hoist_down)
 sonar_running = FALSE;

if (power_on)
{
 if (hoist_down==FALSE)  /* hoist up (sonar not operational) */
  /* hoist up (sonar not operational) */
  {
  /* hoist arrow up */
  move2i(46,-46);
  draw2i(46,-42);
  draw2i(48,-44);
  move2i(46,-42);
  draw2i(44,-44);
  }
 else  /* sonar operational */
  {
  sonar_running = TRUE;

  /* hoist arrow down */
  move2i(46,-42);
  draw2i(46,-46);
  draw2i(44,-44);
  move2i(46,-46);
  draw2i(48,-44);

  boundary_flag = FALSE; /* always set boundary flag false to start */

  if (range_setting_change)
  { /* calculate new beam increment */
   switch (range_setting)
   {
    case 800: *beam_inc = BEAM_INC_800;
       break;
    case 600: *beam_inc = BEAM_INC_600;
       break;
    case 400: *beam_inc = BEAM_INC_400;
       break;
```

```
 case 300: *beam_inc = BEAM_INC_300;
    break;
 case 200: *beam_inc = BEAM_INC_200;
    break;
 case 150: *beam_inc = BEAM_INC_150;
    break;
 case 100: *beam_inc = BEAM_INC_100;
    break;
 case 60: *beam_inc = BEAM_INC_60;
    break;
 case 30: *beam_inc = BEAM_INC_30;
    break;
 case 15: *beam_inc = BEAM_INC_15;
    break;
 } /* end switch */
} /* end if range_setting_change */


/* calculate which video arc is to be updated */
 if (*sweep_location == 0)
  arc_index = 44;
 else if (*sweep_location == 8)
  arc_index = 0;
 else
  arc_index = (*sweep_location / 8) - 1;

/* calculate sins and cos of Euler angles */
trig_calcs(course, dive_angle, roll_angle, tilt_angle,
 *sweep_location, &c4, &c5, &c6, &c7, &c8, &s4, &s5, &s6, &s7, &s8);

/* check to see whether to increment or decrement beam_range */
depth_at_tip = bottom_depth_at_tip(x_coord, y_coord, c4, c5, c6,
 c7, c8, s4, s5, s6, s7, s8, *beam_length);


if (depth_at_tip == BOUNDARY_FLAG_VALUE)
{ /* at a boundary so reset to 0 and increment beam length */
 *beam_length = 0.0;
 *beam_length = find_bottom_inc(&boundary_flag, range_setting, *beam_inc, x_coord,
  y_coord, depth, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, *beam_length) - (0.5 * *beam_inc);
}
else if (depth_at_tip >= tip_depth(depth, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, *beam_length))
{ /* sonar beam tip is above the bottom so increment */
 *beam_length = find_bottom_inc(&boundary_flag, range_setting, *beam_inc, x_coord,
  y_coord, depth, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, *beam_length) - (0.5 * *beam_inc);
}
else
{ /* sonar beam tip is past the bottom so decrement beam length */
 *beam_length = find_bottom_dec(*beam_inc, x_coord, y_coord, depth, c4, c5, c6, c7, c8, s4,
  s5, s6, s7, s8, *beam_length);
 /* sonar beam tip is above the bottom so increment */
 *beam_length = find_bottom_inc(&boundary_flag, range_setting, *beam_inc, x_coord,
  y_coord, depth, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, *beam_length) - (0.5 * *beam_inc);
}
```

```
if (*beam_length > range_setting)
{
 arc_tag = 15; /* target arc is outside of sonar screen */
 *beam_length = range_setting; /* reset beam_length */
}
else
{
 /* convert range to one of 15 arc tags (0 - 14)*/
 arc_tag = round((*beam_length/range_setting) * NUMBER_OF_ARCS);
 /* reset beam_length */
 *beam_length = *beam_length - (0.5 * *beam_inc);
}

if (boundary_flag)
{ /* database boundary reached before tip reached bottom */

 /* no contact color out to boundary distance */
 for (i=0; i<arc_tag; ++i)
 {
 editobj(video_arc_array[arc_index]);
  objreplace(arc_tag_array[arc_index][i]);
  color(NO_CONTACT_COLOR);
 closeobj();
 }
 /* boundary color the rest of the way */
 for (i = arc_tag; i < NUMBER_OF_ARCS; ++i)
 {
 editobj(video_arc_array[arc_index]);
  objreplace(arc_tag_array[arc_index][i]);
  color(BOUNDARY_COLOR);
 closeobj();
 }
}
else  /* no boundary flag */
{
 if (arc_tag < NUMBER_OF_ARCS) /* contact with bottom made */
 {
 *encountered_contact = TRUE;

 for (i=0; i<arc_tag; ++i)
 { /* draw no contact color out to contact range */
 editobj(video_arc_array[arc_index]);
  objreplace(arc_tag_array[arc_index][i]);
  color(NO_CONTACT_COLOR);
 closeobj();
 }
 /* draw contact color */
 editobj(video_arc_array[arc_index]);
 objreplace(arc_tag_array[arc_index][arc_tag]);
 color(CONTACT_COLOR);
 closeobj();
```

```
 if (arc_tag < NUMBER_OF_ARCS_MINUS1)
 { /* draw shadow color rest of way if not at end of beam arc */
  for (i = arc_tag + 1; i < NUMBER_OF_ARCS; ++i)
  {
   editobj(video_arc_array[arc_index]);
    objreplace(arc_tag_array[arc_index][i]);
    color(SHADOW_COLOR);
   closeobj();
  }
 }
}
else  /* no contact made with bottom */
{
 /* draw all no contact color */
 for (i = 0; i < NUMBER_OF_ARCS; ++i)
 {
  editobj(video_arc_array[arc_index]);
   objreplace(arc_tag_array[arc_index][i]);
   color(NO_CONTACT_COLOR);
  closeobj();
 }
} /* else no contact */

} /* else no boundary */

/* call the video return display */
for (i = 0; i < 45; ++i)
 callobj(video_arc_array[i]);

/* call sonar screen objects */
callobj(tilt_angle_indicator);

/* call the sweep line */
callobj(sonar_sweep);

if (sector_setting != 360)
{
 /* call the scan mask */
 callobj(scan_mask);
}

/* make the range ring overlay */
make_range_ring_overlay();

if (range_ring_location > 1)
 /* make the range ring */
 make_range_ring(range_ring_location);

/* call the scan heading line */
callobj(scan_heading_line);

color(WHITE);
/* draw range setting numbers */
if (range_setting == 15)
```

```
{
 draw_numbers(1, RANGE_SETTING_TENS_X, RANGE_SETTING_Y);
 draw_numbers(5, RANGE_SETTING_ONES_X, RANGE_SETTING_Y);
}
else
{
 draw_numbers(0, RANGE_SETTING_ONES_X, RANGE_SETTING_Y);
}

if ((range_setting < 100) && (range_setting !=15))
{
 if (range_setting == 30)
  draw_numbers(3, RANGE_SETTING_TENS_X, RANGE_SETTING_Y);
 else
  draw_numbers(6, RANGE_SETTING_TENS_X, RANGE_SETTING_Y);
 draw_numbers(0, RANGE_SETTING_ONES_X, RANGE_SETTING_Y);
}
else  if (range_setting >= 100)
{
 if (range_setting == 150)
  draw_numbers(5, RANGE_SETTING_TENS_X, RANGE_SETTING_Y);
 else
  draw_numbers(0, RANGE_SETTING_TENS_X, RANGE_SETTING_Y);
 hun_digit = range_setting / 100;
 switch (hun_digit) {
  case 1: draw_numbers(1, RANGE_SETTING_HUN_X, RANGE_SETTING_Y);
    break;
  case 2: draw_numbers(2, RANGE_SETTING_HUN_X, RANGE_SETTING_Y);
    break;
  case 3: draw_numbers(3, RANGE_SETTING_HUN_X, RANGE_SETTING_Y);
    break;
  case 4: draw_numbers(4, RANGE_SETTING_HUN_X, RANGE_SETTING_Y);
    break;
  case 6: draw_numbers(6, RANGE_SETTING_HUN_X, RANGE_SETTING_Y);
    break;
  case 8: draw_numbers(8, RANGE_SETTING_HUN_X, RANGE_SETTING_Y);
    break;
 }
}

/* draw tilt angle numbers */
if (tilt_angle >= 0)
 pos_tilt = tilt_angle;
else
 {
  /* draw tilt - */
  move2(-42.0,40.0);
  draw2(-39.0,40.0);
  pos_tilt = -1 * tilt_angle;
 }

if (pos_tilt > 9)
{
 ones_digit = pos_tilt % 10;
```

```
 tens_digit = pos_tilt / 10;
 }
 else
 {
 ones_digit = pos_tilt;
 tens_digit = 0;
 }
 switch (tens_digit) {
  case 0: draw_numbers(0, TILT_TENS_X, TILT_Y);
   break;
  case 1: draw_numbers(1, TILT_TENS_X, TILT_Y);
   break;
  case 2: draw_numbers(2, TILT_TENS_X, TILT_Y);
   break;
  case 3: draw_numbers(3, TILT_TENS_X, TILT_Y);
   break;
  case 4: draw_numbers(4, TILT_TENS_X, TILT_Y);
   break;
  case 5: draw_numbers(5, TILT_TENS_X, TILT_Y);
   break;
  case 6: draw_numbers(6, TILT_TENS_X, TILT_Y);
   break;
  case 7: draw_numbers(7, TILT_TENS_X, TILT_Y);
   break;
  case 8: draw_numbers(8, TILT_TENS_X, TILT_Y);
   break;
  case 9: draw_numbers(9, TILT_TENS_X, TILT_Y);
   break;
 }

 switch (ones_digit) {
  case 0: draw_numbers(0, TILT_ONES_X, TILT_Y);
   break;
  case 1: draw_numbers(1, TILT_ONES_X, TILT_Y);
   break;
  case 2: draw_numbers(2, TILT_ONES_X, TILT_Y);
   break;
  case 3: draw_numbers(3, TILT_ONES_X, TILT_Y);
   break;
  case 4: draw_numbers(4, TILT_ONES_X, TILT_Y);
   break;
  case 5: draw_numbers(5, TILT_ONES_X, TILT_Y);
   break;
  case 6: draw_numbers(6, TILT_ONES_X, TILT_Y);
   break;
  case 7: draw_numbers(7, TILT_ONES_X, TILT_Y);
   break;
  case 8: draw_numbers(8, TILT_ONES_X, TILT_Y);
   break;
  case 9: draw_numbers(9, TILT_ONES_X, TILT_Y);
   break;
 }

 /* convert tilt to a positive number or zero */
```

```
if (tilt_angle >= 0)
 pos_tilt = 0;
else
 pos_tilt = -1 * tilt_angle;

/* draw range ring location numbers */
temp_range_ring_location = (int)((cos(pos_tilt*DTOR))*
 ((range_ring_location/SONAR_RADIUS)*range_setting));
if (temp_range_ring_location < 10)
{
 ones_digit = temp_range_ring_location;
 tens_digit = 0;
 hun_digit = 0;
}
else if (temp_range_ring_location < 100)
{
 ones_digit = temp_range_ring_location % 10;
 tens_digit = temp_range_ring_location / 10;
 hun_digit = 0;
}
else
{
 ones_digit = temp_range_ring_location % 10;
 tens_digit = (temp_range_ring_location / 10) % 10;
 hun_digit = temp_range_ring_location / 100;
}
switch (hun_digit) {
 case 0: draw_numbers(0, RANGE_RING_HUN_X, RANGE_RING_Y);
  break;
 case 1: draw_numbers(1, RANGE_RING_HUN_X, RANGE_RING_Y);
  break;
 case 2: draw_numbers(2, RANGE_RING_HUN_X, RANGE_RING_Y);
  break;
 case 3: draw_numbers(3, RANGE_RING_HUN_X, RANGE_RING_Y);
  break;
 case 4: draw_numbers(4, RANGE_RING_HUN_X, RANGE_RING_Y);
  break;
 case 5: draw_numbers(5, RANGE_RING_HUN_X, RANGE_RING_Y);
  break;
 case 6: draw_numbers(6, RANGE_RING_HUN_X, RANGE_RING_Y);
  break;
 case 7: draw_numbers(7, RANGE_RING_HUN_X, RANGE_RING_Y);
  break;
 case 8: draw_numbers(8, RANGE_RING_HUN_X, RANGE_RING_Y);
  break;
 case 9: draw_numbers(9, RANGE_RING_HUN_X, RANGE_RING_Y);
  break;
}

switch (tens_digit) {
 case 0: draw_numbers(0, RANGE_RING_TENS_X, RANGE_RING_Y);
  break;
 case 1: draw_numbers(1, RANGE_RING_TENS_X, RANGE_RING_Y);
  break;
```

```
 case 2: draw_numbers(2, RANGE_RING_TENS_X, RANGE_RING_Y);
  break;
 case 3: draw_numbers(3, RANGE_RING_TENS_X, RANGE_RING_Y);
  break;
 case 4: draw_numbers(4, RANGE_RING_TENS_X, RANGE_RING_Y);
  break;
 case 5: draw_numbers(5, RANGE_RING_TENS_X, RANGE_RING_Y);
  break;
 case 6: draw_numbers(6, RANGE_RING_TENS_X, RANGE_RING_Y);
  break;
 case 7: draw_numbers(7, RANGE_RING_TENS_X, RANGE_RING_Y);
  break;
 case 8: draw_numbers(8, RANGE_RING_TENS_X, RANGE_RING_Y);
  break;
 case 9: draw_numbers(9, RANGE_RING_TENS_X, RANGE_RING_Y);
  break;
}

switch (ones_digit) {
 case 0: draw_numbers(0, RANGE_RING_ONES_X, RANGE_RING_Y);
  break;
 case 1: draw_numbers(1, RANGE_RING_ONES_X, RANGE_RING_Y);
  break;
 case 2: draw_numbers(2, RANGE_RING_ONES_X, RANGE_RING_Y);
  break;
 case 3: draw_numbers(3, RANGE_RING_ONES_X, RANGE_RING_Y);
  break;
 case 4: draw_numbers(4, RANGE_RING_ONES_X, RANGE_RING_Y);
  break;
 case 5: draw_numbers(5, RANGE_RING_ONES_X, RANGE_RING_Y);
  break;
 case 6: draw_numbers(6, RANGE_RING_ONES_X, RANGE_RING_Y);
  break;
 case 7: draw_numbers(7, RANGE_RING_ONES_X, RANGE_RING_Y);
  break;
 case 8: draw_numbers(8, RANGE_RING_ONES_X, RANGE_RING_Y);
  break;
 case 9: draw_numbers(9, RANGE_RING_ONES_X, RANGE_RING_Y);
  break;
}

/* draw depth numbers */
depth_value = (int)((sin(pos_tilt*DTOR))* ((range_ring_location/SONAR_RADIUS)*
 range_setting));
if (depth_value < 10)
{
 ones_digit = depth_value;
 tens_digit = 0;
 hun_digit = 0;
}
else if (depth_value < 100)
{
 ones_digit = depth_value % 10;
 tens_digit = depth_value / 10;
```

```
 hun_digit = 0;
}
else
{
 ones_digit = (depth_value % 10);
 tens_digit = (depth_value / 10) % 10;
 hun_digit = depth_value / 100;
}
switch (hun_digit) {
 case 0: draw_numbers(0, DEPTH_HUN_X, DEPTH_Y);
  break;
 case 1: draw_numbers(1, DEPTH_HUN_X, DEPTH_Y);
  break;
 case 2: draw_numbers(2, DEPTH_HUN_X, DEPTH_Y);
  break;
 case 3: draw_numbers(3, DEPTH_HUN_X, DEPTH_Y);
  break;
 case 4: draw_numbers(4, DEPTH_HUN_X, DEPTH_Y);
  break;
 case 5: draw_numbers(5, DEPTH_HUN_X, DEPTH_Y);
  break;
 case 6: draw_numbers(6, DEPTH_HUN_X, DEPTH_Y);
  break;
 case 7: draw_numbers(7, DEPTH_HUN_X, DEPTH_Y);
  break;
 case 8: draw_numbers(8, DEPTH_HUN_X, DEPTH_Y);
  break;
 case 9: draw_numbers(9, DEPTH_HUN_X, DEPTH_Y);
  break;
}

switch (tens_digit) {
 case 0: draw_numbers(0, DEPTH_TENS_X, DEPTH_Y);
  break;
 case 1: draw_numbers(1, DEPTH_TENS_X, DEPTH_Y);
  break;
 case 2: draw_numbers(2, DEPTH_TENS_X, DEPTH_Y);
  break;
 case 3: draw_numbers(3, DEPTH_TENS_X, DEPTH_Y);
  break;
 case 4: draw_numbers(4, DEPTH_TENS_X, DEPTH_Y);
  break;
 case 5: draw_numbers(5, DEPTH_TENS_X, DEPTH_Y);
  break;
 case 6: draw_numbers(6, DEPTH_TENS_X, DEPTH_Y);
  break;
 case 7: draw_numbers(7, DEPTH_TENS_X, DEPTH_Y);
  break;
 case 8: draw_numbers(8, DEPTH_TENS_X, DEPTH_Y);
  break;
 case 9: draw_numbers(9, DEPTH_TENS_X, DEPTH_Y);
  break;
}
```

```
switch (ones_digit) {
 case 0: draw_numbers(0, DEPTH_ONES_X, DEPTH_Y);
  break;
 case 1: draw_numbers(1, DEPTH_ONES_X, DEPTH_Y);
  break;
 case 2: draw_numbers(2, DEPTH_ONES_X, DEPTH_Y);
  break;
 case 3: draw_numbers(3, DEPTH_ONES_X, DEPTH_Y);
  break;
 case 4: draw_numbers(4, DEPTH_ONES_X, DEPTH_Y);
  break;
 case 5: draw_numbers(5, DEPTH_ONES_X, DEPTH_Y);
  break;
 case 6: draw_numbers(6, DEPTH_ONES_X, DEPTH_Y);
  break;
 case 7: draw_numbers(7, DEPTH_ONES_X, DEPTH_Y);
  break;
 case 8: draw_numbers(8, DEPTH_ONES_X, DEPTH_Y);
  break;
 case 9: draw_numbers(9, DEPTH_ONES_X, DEPTH_Y);
  break;
 } /* switch */

 } /* else sonar operational */

 cmov2i(-46,-42);
 charstr("SS265");
 cmov2i(-46,-46);
 charstr("WESMAR");
 cmov2i(26,-46);
 charstr("HOIST");

 /* draw range setting R */
 move2(-46.0, 44.0);
 draw2(-46.0,46.0);
 rect(-46.0,46.0,-42.2,48.0);
 move2(-44.0,46.0);
 draw2(-42.0,44.0);

 /* draw tilt T */
 move2(-44.0,38.0);
 draw2(-44.0,42.0);
 move2(-46.0,42.0);
 draw2(-42.0,42.0);

 /* draw tilt degree circle */
 circ(-26.0,41.5,0.5);

 /* draw range setting M */
 move2(-22.0,44.0);
 draw2(-21.0,48.0);
 draw2(-20.0,44.0);
 draw2(-19.0,48.0);
 draw2(-18.0,44.0);
```

```
/* draw range ring M */
move2(44.0,44.0);
draw2(45.0,48.0);
draw2(46.0,44.0);
draw2(47.0,48.0);
draw2(48.0,44.0);

/* draw depth M */
move2(44.0,38.0);
draw2(45.0,42.0);
draw2(46.0,38.0);
draw2(47.0,42.0);
draw2(48.0,38.0);

/* draw range ring arrow */
move2(20.0,46.0);
draw2(26.0,46.0);
draw2(24.0,48.0);
move2(26.0,46.0);
draw2(24.0,44.0);

/* draw depth arrow */
move2(46.0,36.0);
draw2(46.0,30.0);
draw2(44.0,34.0);
move2(46.0,30.0);
draw2(48.0,34.0);

/* logo color bar */
color(RED);
rectf(42.0,-25.0,48.0,-22.0);
color(YELLOW);
rectf(42.0,-28.0,48.0,-25.0);
color(GREEN);
rectf(42.0,-31.0,48.0,-28.0);
color(CYAN);
rectf(42.0,-34.0,48.0,-31.0);
color(MAGENTA);
rectf(42.0,-37.0,48.0,-34.0);
color(BLUE);
rectf(42.0,-40.0,48.0,-37.0);

/* OUTSIDE CIRCLE FOR SONAR */
linewidth(3);
color(WHITE);
circ(0.0,0.0,SONAR_RADIUS+ 0.2);

} /* power on */

} /* end make_video.c */


/* function to find the row and col indices depending on x and y
```

```
location of the sub and extract the value from the depth data base */
int bottom_depth_at_tip(x_coord, y_coord, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length)

float x_coord, y_coord, c4, c5, c6 ,c7, c8, s4, s5, s6, s7, s8, beam_length;
{
 extern int bottom_data[BOTTOM_POINTS_WIDTH][BOTTOM_POINTS_HEIGHT];
 int row_index, col_index;
 float a09_2_4(), a09_1_4();
 int value;

 row_index = (int)(-1 * a09_2_4(y_coord, c4, c5, c6, c7, c8, s4, s5, s6,
  s7, s8, beam_length)/RESOLUTION_IN_METERS);
 col_index = (int)(a09_1_4(x_coord, c4, c5, c6, c7, c8, s4, s5, s6,
  s7, s8, beam_length)/RESOLUTION_IN_METERS);

 if ((row_index < 0) || (row_index > BOTTOM_POINTS_HEIGHT-1) ||
  (col_index < 0) || (col_index > BOTTOM_POINTS_WIDTH-1))
 {  /* out of database bounds */
  return BOUNDARY_FLAG_VALUE;
 }
 else
 {
  value = bottom_data[row_index][col_index];
  return value;
 }

} /* end bottom_depth_at_tip  */


/* function to increment the beam length until it hits bottom or boundary */
float find_bottom_inc(boundary_flag, range_setting, beam_inc, x_coord, y_coord, depth, c4, c5, c6,
 c7, c8, s4, s5, s6, s7, s8, beam_length)

float beam_inc, x_coord, y_coord, depth, c4, c5, c6 ,c7, c8, s4, s5, s6, s7, s8, beam_length;
short *boundary_flag;
int range_setting;

{
 int depth_at_tip;
 int bottom_depth_at_tip();
 float tip_depth();

 do
 {
  beam_length = beam_length + beam_inc;
  depth_at_tip = bottom_depth_at_tip(x_coord, y_coord, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8,
   beam_length);
 } while ((depth_at_tip > tip_depth(depth, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length)) &&
  (beam_length <= range_setting) && (depth_at_tip != BOUNDARY_FLAG_VALUE));

 if ((depth_at_tip == BOUNDARY_FLAG_VALUE) || (beam_length > range_setting))
  *boundary_flag = TRUE;

 return beam_length;
```

```
} /* end find_bottom_inc */


/* function to decrement the beam length until tip depth is less than bottom */
float find_bottom_dec(beam_inc, x_coord, y_coord, depth, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8,
 beam_length)

float beam_inc, x_coord, y_coord, depth, c4, c5, c6 ,c7, c8, s4, s5, s6, s7, s8, beam_length;

{
 int bottom_depth_at_tip();
 float tip_depth();

 do
 {
  beam_length = beam_length - beam_inc;
 } while (tip_depth(depth, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length) >
  bottom_depth_at_tip(x_coord, y_coord, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length));

 return beam_length;
} /* end find_bottom_dec */


/* calculate the depth of the beam tip */
float tip_depth(depth, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length)

float depth, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length;

{
float a09_3_4();
float value;

 value = FEET_PER_M * a09_3_4(depth, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length);
 return value;
} /* end tip_depth */


/* calculate the sins/cos of Euler angles */
trig_calcs(course, sub_dive_angle, sub_roll, tilt_angle,
 sweep_location, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8)

float *c4, *c5, *c6, *c7, *c8, *s4, *s5, *s6, *s7, *s8;
int course, sub_dive_angle, sub_roll, tilt_angle, sweep_location;

{
 *c4 = cos (DTOR * course);
 *s4 = sin (DTOR * course);
 *c5 = cos (DTOR * sub_dive_angle);
 *s5 = sin (DTOR * sub_dive_angle);
 *c6 = cos (DTOR * sub_roll);
 *s6 = sin (DTOR * sub_roll);
 *c7 = cos (DTOR * tilt_angle);
 *s7 = sin (DTOR * tilt_angle);
 *c8 = cos (DTOR * sweep_location);
```

```
*s8 = sin (DTOR * sweep_location);
} /* end trig_calcs */


/* x_coord of beam tip in base coord system */
float a09_1_4(x_coord, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length)

float x_coord, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length;

{
 float value;

  return (x_coord
   + (c4 * c5 * beam_length * c7 * c8)
   + ((beam_length * c7 * s8) * ((c4 * s5 * s6) - (s4 * c6)))
   - ((beam_length * s7) * ((c4 * s5 * c6) + (s4 * s6))));

} /* end a09_1_4 */


/* y_coord of beam tip in base coord system */
float a09_2_4(y_coord, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length)

float y_coord, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length;

{
 float value;

  return (y_coord
   + (s4 * c5 * beam_length * c7 * c8)
   + ((beam_length * c7 * s8)*((c4 * c6) + (s4 * s5 * s6)))
   - ((beam_length * s7) * ((s4 * s5 * c6) - (c4 * s6))));

} /* end a09_2_4 */


/* depth of beam tip in base coord system */
float a09_3_4(depth, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8,beam_length)

float depth, c4, c5, c6, c7, c8, s4, s5, s6, s7, s8, beam_length;

{
 float value;

  return (depth - (s5 * beam_length * c7 * c8)
   + (c5 * s6 * beam_length * c7 * s8)
   - (c5 * c6 * beam_length * s7));

} /* end a09_3_4 */
```

```
/*+-------------------------------+
 |
 |      read_ctrls.c
 |
 +-------------------------------+              */

/* This procedure is called by main_sonar to read the values
 from the operator's controls (mouse, dials and keyboard) */

#include "gl.h"              /* graphics lib defs */
#include "sonar.h"             /* sonar constants */
#include "device.h"          /* device definitions */

read_controls(scan_mode, power_dial_change, volume_dial_change, sector_dial_change,
 range_dial_change, sector_setting_change, tilt_angle_change, scan_heading_change,
 previous_sector_setting, previous_range_setting, power_on, hoist_down, range_setting_change,
 power_dial_location, volume_dial_location, sector_dial_location, sector_setting,
 range_dial_location, range_setting, tilt_angle, tilt_inc,  scan_heading,
 range_ring_location, scan_toggle_up, tilt_toggle_up, range_toggle_up, exit, selected_course,
 selected_speed, selected_dive_angle, set, drift, change_location,
 request_depth_array_display)

 short *scan_mode, *power_dial_change, *volume_dial_change, *sector_dial_change,
  *range_dial_change,
  *sector_setting_change, *tilt_angle_change, *scan_heading_change, *power_on, *hoist_down,
  *range_setting_change, *scan_toggle_up, *tilt_toggle_up, *range_toggle_up, *exit,
  *change_location, *request_depth_array_display;
 int *previous_sector_setting, *previous_range_setting, *power_dial_location,
  *volume_dial_location, *sector_dial_location, *sector_setting, *range_dial_location,
  *range_setting, *tilt_angle, *tilt_inc, *scan_heading, *range_ring_location,
  *selected_course, *selected_dive_angle, *set;
 float *selected_speed, *drift;

 {
 extern int bottom_data[BOTTOM_POINTS_HEIGHT][BOTTOM_POINTS_WIDTH];
 float x,y, y_ortho;
 int inside();

 /* quit if EKEY is pressed */
 if(getbutton(EKEY))
  *exit = TRUE;
 else
 {
  if (getbutton(SKEY))
   *request_depth_array_display = TRUE;

  if (getbutton(ZEROKEY))
   *scan_mode = JUST_SCAN_NO_STORE;

  if (getbutton(ONEKEY))
   *scan_mode = ONE_SCAN_AND_STORE;

  if (getbutton(TWOKEY))
   *scan_mode = COMPLETE_SCAN_AND_STORE;
```

```
if (getbutton(UPARROWKEY))
{
 *tilt_inc = *tilt_inc + 1;
 if (*tilt_inc > 90)
  *tilt_inc = 90;
}

if (getbutton(DOWNARROWKEY))
{
 *tilt_inc = *tilt_inc - 1;
 if (*tilt_inc < 0)
  *tilt_inc = 0;
}

if (getvaluator(DIAL1) != *volume_dial_location)
{ /* change in volume dial */
 *volume_dial_change = TRUE;
 *volume_dial_location = getvaluator(DIAL1);
 if (*volume_dial_location < VOLUME_OFF)
   *hoist_down = FALSE;
 else
   *hoist_down = TRUE;
}

if (getvaluator(DIAL0) != *power_dial_location)
{ /* change in power dial */
 *power_dial_change = TRUE;
 *power_dial_location = getvaluator(DIAL0);
 if (*power_dial_location < POWER_OFF)
 {  *power_on = FALSE; }
 else
 {  *power_on = TRUE; }
}

if (getvaluator(DIAL2) != *sector_dial_location)
{ /* change in sector dial */
 *sector_dial_change = TRUE;
 *sector_dial_location = 280 - getvaluator(DIAL2);
 if (*sector_dial_location <= 35) *sector_setting = 360;
 else if ((*sector_dial_location > 35) &&
    (*sector_dial_location <= 70)) *sector_setting = 315;
 else if ((*sector_dial_location > 70) &&
    (*sector_dial_location <= 105)) *sector_setting = 180;
 else if ((*sector_dial_location > 105) &&
    (*sector_dial_location <= 140)) *sector_setting = 135;
 else if ((*sector_dial_location > 140) &&
    (*sector_dial_location <= 175)) *sector_setting = 90;
 else if ((*sector_dial_location > 175) &&
    (*sector_dial_location <= 210)) *sector_setting = 45;
 else if ((*sector_dial_location > 210) &&
    (*sector_dial_location <= 245)) *sector_setting = 30;
 else  *sector_setting = 15;
```

```
 if (*previous_sector_setting != *sector_setting)
 {
  *previous_sector_setting = *sector_setting;
  *sector_setting_change = TRUE;
 }
} /* sector dial */

if (getvaluator(DIAL3) != *range_dial_location)
{ /* change in range dial */
 *range_dial_change = TRUE;
 *range_dial_location = 300 - getvaluator(DIAL3);
 if (*range_dial_location <= 30) *range_setting = 800;
 else if ((*range_dial_location > 30) &&
    (*range_dial_location <= 60)) *range_setting = 600;
 else if ((*range_dial_location > 60) &&
    (*range_dial_location <= 90)) *range_setting = 400;
 else if ((*range_dial_location > 90) &&
    (*range_dial_location <= 120)) *range_setting = 300;
 else if ((*range_dial_location > 120) &&
    (*range_dial_location <= 150)) *range_setting = 200;
 else if ((*range_dial_location > 150) &&
    (*range_dial_location <= 180)) *range_setting = 150;
 else if ((*range_dial_location > 180) &&
    (*range_dial_location <= 210)) *range_setting = 100;
 else if ((*range_dial_location > 210) &&
    (*range_dial_location <= 240)) *range_setting = 60;
 else if ((*range_dial_location > 240) &&
    (*range_dial_location <= 270)) *range_setting = 30;
 else  *range_setting = 15;

 if (*previous_range_setting != *range_setting)
 {
  *previous_range_setting = *range_setting;
  *range_setting_change = TRUE;
 }

} /* range_dial */

if (getvaluator(DIAL4) != *selected_course)
{ /* change in course dial */
 *selected_course = getvaluator(DIAL4);
}

if (getvaluator(DIAL5) != *set) /* change in set dial */
{
 *set = getvaluator(DIAL5);
}

if (getvaluator(DIAL6) != *selected_dive_angle)
{ /*change in dive_angle dial*/
 *selected_dive_angle = getvaluator(DIAL6);
}

/* if middle mouse button hit, check for pick box */
```

```
if (getbutton(MOUSE2))  /* read x and y from mouse */
{
 x = getvaluator(MOUSEX);
 y = getvaluator(MOUSEY);

 /* do we have a tilt toggle hit? */
 if (inside(x,y,14.0,18.0,4.0,8.0,0.0,100.0,0.0,14.0, 0,1073,0,120)) /* hit on toggle */
 { *tilt_toggle_up = !(*tilt_toggle_up);
 }

 /* do we have a scan toggle hit? */
 if (inside(x,y,22.0,26.0,4.0,8.0,0.0,100.0,0.0,14.0, 0,1073,0,120)) /* hit on toggle */
 { *scan_toggle_up = !(*scan_toggle_up);
 }

 /* do we have a range toggle hit? */
 if (inside(x,y,30.0,34.0,4.0,8.0,0.0,100.0,0.0,14.0, 0,1073,0,120)) /* hit on toggle */
 { *range_toggle_up = !*range_toggle_up;
 }

 /* do we have a speed bar hit? */
 if (inside(x,y,29.75,36.25,53.0,71.0,0.0,100.0,0.0,74.0, 647,1023,120,391)) /* hit on speed bar */
 {
  /* convert y to ortho coords */
  /* y_ortho = ((y - vminy)*(omaxy - ominy))/(vmaxy - vminy); */
  y_ortho = ((y-120)*74.0)/271;
  /* ortho bar range = omaxy - ominy = 18 */
  /* convert y_ortho to auv speed */
  /* *selected_speed = ((MAX_SPEED - MIN_SPEED)*(y_ortho - ZERO_Y))/ortho bar range */
  *selected_speed = ((MAX_SPEED - MIN_SPEED) * (y_ortho - ZERO_Y))/18;
 }

 /* do we have a drift bar hit? */
 if (inside(x,y,46.0,53.0,16.0,34.0,0.0,100.0,0.0,74.0, 647,1023,120,391)) /* hit on drift bar */
 {
  /* convert y to ortho coords */
  /* y_ortho = ((y - vminy)*(omaxy - ominy))/(vmaxy - vminy) */
  y_ortho = ((y-120)*74.0)/271;

  /* ortho bar range = omaxy - ominy = 18 */
  /* convert y_ortho to current drift */
  /* *drift = ((y_ortho - ominy)* MAX_DRIFT)/ortho_bar_range*/
  *drift = ((y_ortho - 16)*MAX_DRIFT)/18;
 }

} /* MOUSE2 */

/* if MOUSE1, increment to toggle values selected */
if (getbutton(MOUSE1))  /* read x and y from mouse */
{
 x = getvaluator(MOUSEX);
 y = getvaluator(MOUSEY);

 /* do we have a tilt toggle hit? */
```

```
 if (inside(x,y,14.0,18.0,4.0,8.0,0.0,100.0,0.0,14.0, 0,1073,0,120)) /* hit on toggle */
 {
  if (*tilt_toggle_up) *tilt_angle += 1;
  else *tilt_angle -= 1;
  if (*tilt_angle < -90) *tilt_angle = -90;
  if (*tilt_angle > 4) *tilt_angle = 4;
    *tilt_angle_change = TRUE;
 }  /* tilt toggle increment */

 /* do we have a scan toggle hit? */
 if (inside(x,y,22.0,26.0,4.0,8.0,0.0,100.0,0.0,14.0, 0,1073,0,120)) /* hit on toggle */
 {
  if (*scan_toggle_up) *scan_heading += 1;
  else *scan_heading -= 1;
  if (*scan_heading < 0) *scan_heading = 360 + *scan_heading;
  if (*scan_heading > 360) *scan_heading = 360 - *scan_heading;
    *scan_heading_change = TRUE;
 }  /* scan toggle increment */

 /* do we have a range toggle hit? */
 if (inside(x,y,30.0,34.0,4.0,8.0,0.0,100.0,0.0,14.0, 0,1073,0,120)) /* hit on toggle */
 {
  if (*range_toggle_up) *range_ring_location -= 1;
  else *range_ring_location += 1;
  if (*range_ring_location < 0)
   *range_ring_location = 0;
  if (*range_ring_location > SONAR_RADIUS)
   *range_ring_location = SONAR_RADIUS;
 }  /* range toggle increment */

 } /* MOUSE1 */

 /* if MOUSE3, change the auv's position/depth */
 if (getbutton(MOUSE3))
  *change_location = TRUE;

 } /* not exit */

} /* read_controls */
```

```
/*+-------------------------------+
 |
 |      read_sets.c
 |
 +-------------------------------+              */

/* This procedure is called by get_auv_settings to read the values from the operator's controls
   (mouse and dials)when a change in location/depth is requested */

#include "gl.h"              /* graphics lib defs */
#include "sonar.h"             /* sonar constants */
#include "device.h"           /* device definitions */
#include "math.h"            /* math definitions */

read_settings(scan_mode, x_coord, y_coord, depth, course, speed, dive_angle, selected_course,
selected_speed, selected_dive_angle, set, drift, cog, sog, tilt_inc, done)

int *course, *dive_angle, *set, *cog, *selected_course, *selected_dive_angle, *tilt_inc;
float *x_coord, *y_coord, *depth, *speed, *drift, *sog, *selected_speed;
short *scan_mode, *done;

{
extern int bottom_data[BOTTOM_POINTS_HEIGHT][BOTTOM_POINTS_WIDTH];
int inside();
int water_depth;
float x,y, y_ortho, course_x, course_y, current_x, current_y, cog_x, cog_y;

/* quit if left mouse button is pushed */
if(getbutton(MOUSE3))
 *done = TRUE;
else
{
 if (getbutton (ZEROKEY))
  *scan_mode = JUST_SCAN_NO_STORE;

 if (getbutton (ONEKEY))
  *scan_mode = ONE_SCAN_AND_STORE;

 if (getbutton (TWOKEY))
  *scan_mode = COMPLETE_SCAN_AND_STORE;

 if (getbutton (UPARROWKEY))
 {
  *tilt_inc = *tilt_inc + 1;
  if (*tilt_inc > 90)
   *tilt_inc = 90;
 }

 if (getbutton (DOWNARROWKEY))
 {
  *tilt_inc = *tilt_inc - 1;
  if (*tilt_inc < 90)
   *tilt_inc = 0;
 }
```

```
if (getvaluator(DIAL4) != *course) /* change in course dial */
{
 *course = getvaluator(DIAL4);
}

if (getvaluator(DIAL5) != *set) /* change in set dial */
{
 *set = getvaluator(DIAL5);
}

if (getvaluator(DIAL6) != *dive_angle) /*change in dive_angle dial*/
{
 *dive_angle = getvaluator(DIAL6);
}

/* if middle mouse button hit, check for pick box */
if (getbutton(MOUSE2))  /* read x and y from mouse */
{
 x = getvaluator(MOUSEX);
 y = getvaluator(MOUSEY);

 /* do we have a chart hit? */
 if (inside(x,y,0.0,1000.0,0.0,1000.0,0.0,1000.0,0.0,1000.0, 0,647,120,767)) /* hit on chart */
 {
  /* convert x, y to ortho coords */
  /* *x_coord = ((x - vminx) * (omaxx - ominx))/(vmaxx - vminx)*/
  *x_coord = (x * 1000.0)/647;
  /* *y_coord = -1 * ((y - vminy) * (omaxy - ominy))/(vmaxy- vminy)*/
  *y_coord = -1 * (((y-120) * 1000.0)/647);
 }

 /* do we have a speed bar hit? */
 if (inside(x,y,29.75,36.25,53.0,71.0,0.0,100.0,0.0,74.0, 647,1023,120,391)) /* hit on speed bar */
 {
  /* convert y to ortho coords */
  /* y_ortho = ((y - vminy)*(omaxy - ominy))/(vmaxy - vminy) */
  y_ortho = ((y-120)*74.0)/271;
  /* ortho bar range = omaxy - ominy = 18 */
  /* convert y_ortho to auv speed */
  /* *speed = ((MAX_SPEED - MIN_SPEED) *
   (y_ortho - ZERO_Y))/ortho bar range */
  *speed = ((MAX_SPEED - MIN_SPEED) * (y_ortho - ZERO_Y))/18;
 }

 /* do we have a drift bar hit? */
 if (inside(x,y,46.0,53.0,16.0,34.0,0.0,100.0,0.0,74.0, 647,1023,120,391)) /* hit on drift bar */
 {
  /* convert y to ortho coords */
  /* y_ortho = ((y - vminy)*(omaxy - ominy))/(vmaxy - vminy) */
  y_ortho = ((y-120)*74.0)/271;

  /* ortho bar range = omaxy - ominy = 18 */
  /* convert y_ortho to current drift */
```

```
 /* *drift = ((y_ortho - ominy)* MAX_DRIFT)/ortho_bar_range*/
 *drift = ((y_ortho - 16)*MAX_DRIFT)/18;
 }

 /* do we have a depth bar hit? */
 if (inside(x,y,27.0,33.0,3.0,30.0,0.0,100.0,0.0,33.0, 647,1023,0,120)) /* hit on depth bar */
 {
  /* convert y to ortho coords */
  /* v_ortho = ((y - vminy)*(omaxy - ominy))/(vmaxy - vminy) */
  y_ortho = (y*27.0)/120;

  /* get water depth at current location */
  water_depth = bottom_data[round((-1 * *y_coord) /
   RESOLUTION_IN_METERS)][round(*x_coord/RESOLUTION_IN_METERS)];
  /* ortho bar range = omaxy - ominy */
  /* convert y_ortho to auv depth */
  /* *depth = (water_depth * (y_ortho - omaxy))/orhto_bar_range*/
  /* in feet */
  *depth = (water_depth * (30 - y_ortho))/27;
  if (*depth > water_depth)
   *depth = (float)water_depth;
  /* in meters */
  *depth = M_PER_FEET * *depth;
 }

 } /* MOUSE2 */

 /* calc cog, sog */
 course_x = sin(*course * DTOR) * *speed;
 course_y = cos(*course * DTOR) * *speed;
 current_x = sin(*set * DTOR) * *drift;
 current_y = cos(*set * DTOR) * *drift;
 cog_x = course_x + current_x;
 cog_y = course_y + current_y;
 /* 9 cases */
 if ((cog_x == 0) && (cog_y == 0))
  *cog = *course;
 else if ((cog_x == 0) && (cog_y > 0))
  *cog = 0;
 else if ((cog_x == 0) && (cog_y < 0))
  *cog = 180;
 else if ((cog_y == 0) && (cog_x < 0))
  *cog = 270;
 else if ((cog_y == 0) && (cog_x > 0))
  *cog = 90;
 else if ((cog_x < 0) && (cog_y > 0)) /* 270 - 360 */
  *cog = 270 + (int)(atan(cog_y/(-1 * cog_x)) * RTOD);
 else if ((cog_x < 0) && (cog_y < 0)) /* 180 - 270 */
  *cog = 270 - (int)(atan(cog_y/cog_x) * RTOD);
 else if ((cog_x > 0) && (cog_y > 0)) /* 0 - 90 */
  *cog = 90 - (int)(atan(cog_y/cog_x) * RTOD);
 else if ((cog_x > 0) && (cog_y < 0)) /* 90 - 180 */
  *cog = 90 + (int)(atan((-1 * cog_y)/cog_x) * RTOD);
```

```
 *sog = sqrt((cog_x * cog_x) + (cog_y * cog_y));

 /* reset selected vars */
 *selected_course = *course;
 *selected_speed = *speed:
 *selected_dive_angle = *dive_angle;

 } /* not exit */

} /* read_settings */



/* this function determines if (x,y) is inside the box
defined by the coordinates (xmin,ymin)-(xmax,ymax) */

int inside(x,y,xmin,xmax,ymin,ymax,ominx,omaxx,ominy,omaxy,vminx,vmaxx,vminy,vmaxy)

float x,y,xmin,ymin,xmax,ymax,ominx,omaxx,ominy,omaxy;
int vminx,vmaxx,vminy,vmaxy;

{
 /* convert to world coords */
 xmin=(((vmaxx - vminx)/(omaxx - ominx)) * (xmin - ominx)) + vminx;
 xmax=(((vmaxx - vminx)/(omaxx - ominx)) * (xmax - ominx)) + vminx;
 ymin=(((vmaxy - vminy)/(omaxy - ominy)) * (ymin - ominy)) + vminy;
 ymax=(((vmaxy - vminy)/(omaxy - ominy)) * (ymax - ominy)) + vminy;

 if ((xmin <= x) && (x <= xmax) && (ymin <= y) && (y <= ymax))
 {
  return(TRUE);
 }
 else
 {
  return(FALSE);
 }

} /* inside */
```

```
/*+-------------------------------+
 |
 |    reset_flags.c
 |
 +-------------------------------+              */

/* reset_flags - this procedure is called by main_sonar to reset all the change flags */

#include "gl.h"
#include "sonar.h"

reset_flags(scan_complete, power_dial_change, volume_dial_change, sector_dial_change,
 range_dial_change, scan_heading_change, sector_setting_change, range_setting_change,
 change_location, encountered_contact, request_depth_array_display)

 short *scan_complete, *power_dial_change, *volume_dial_change, *sector_dial_change,
  *range_dial_change, *change_location, *scan_heading_change, *sector_setting_change,
  *range_setting_change, *encountered_contact, *request_depth_array_display;

{
 *scan_complete = FALSE;
 *power_dial_change = FALSE;
 *volume_dial_change = FALSE;
 *sector_dial_change = FALSE;
 *range_dial_change = FALSE;
 *scan_heading_change = FALSE;
 *sector_setting_change = FALSE;
 *range_setting_change = FALSE;
 *change_location = FALSE;
 *encountered_contact = FALSE;
 *request_depth_array_display = FALSE;

} /* reset_flags */
```

```
/*+-------------------------------+
 |
 |      scan_ctrl.c
 |
 +-------------------------------+              */

/* This procedure is called by main_sonar to determine
 when a scan has completed depending upon the scan mode */

#include "sonar.h"
#include "gl.h"

bottom_scan_controller(scan_mode, power_on, hoist_down, tilt_angle, tilt_angle_change, tilt_inc,
 sweep_location, scan_complete)

 short scan_mode, power_on, hoist_down, *tilt_angle_change, *scan_complete;
 int *tilt_angle, tilt_inc, sweep_location;

{
 if (scan_mode != JUST_SCAN_NO_STORE)
 { /* must be one of other 2 scan modes */
  if (power_on && hoist_down && (sweep_location == 0))
  { /* sonar is operational and is at the end of a 360 deg. sweep */
   /* determine scan mode */
   if (scan_mode == ONE_SCAN_AND_STORE)
    *scan_complete = TRUE;
   else
   { /* scan mode = COMPLETE_SCAN_AND_STORE */
    /* decrement tilt_angle */
    *tilt_angle = *tilt_angle - tilt_inc;
    *tilt_angle_change = TRUE;
    if (*tilt_angle < -90)
    {
     *scan_complete = TRUE;
     *tilt_angle = 0; /* reset to 0 */
    }
   } /* end else */
  } /* end if */
 } /* end if not JUST_SCAN */

} /* end bottom_scan_controller */
```

```
/*+-------------------------------+
 |
 |   store_data.c
 |
 +-------------------------------+              */

/* Stores the information received from the sonar in an array
 so regression analysis may be performed. */

#include "sonar.h"
#include "math.h"
#include "gl.h"

store_return_data(scan_mode, encountered_contact, auv_depth, sweep_location,beam_length,
 beam_inc, tilt_angle, left_depth_array_active, left_array_max_depth, left_array_min_depth,
 right_array_max_depth, right_array_min_depth)

 short scan_mode, encountered_contact, left_depth_array_active;
 int sweep_location, tilt_angle;
 float auv_depth, beam_length, beam_inc;
 int *left_array_max_depth, *left_array_min_depth, *right_array_max_depth,
  *right_array_min_depth;

{
 int x_cart, y_cart, depth_cart; /* cartesian conversion vars */
 int x_index, y_index; /* depth array indicies from cartesian coords */
 extern int left_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
 extern int right_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
 extern int min_depth, max_depth;

 if (scan_mode != JUST_SCAN_NO_STORE)
 { /* must be one of other two scans */
  if (encountered_contact)
  {  /* convert the contact location from polar to cart. coords */
   x_cart = (int)((sin(sweep_location * DTOR) *
    cos(tilt_angle * DTOR)* beam_length)/beam_inc);
   y_cart = (int)((cos(sweep_location * DTOR) *
    cos(tilt_angle * DTOR) * beam_length)/beam_inc);
   /* the depth calculated is the sub depth plus the depth of
    the contact below the sub (therefore the actual depth of the
    contact) */
   /* depth is calculated in meters but stored in feet */
   depth_cart = (int)(((sin(-tilt_angle * DTOR) * beam_length) +
    auv_depth) * FEET_PER_M);

   /* check for depth calculations that are greater than
    max depth or less than the min depth of the database */
   if (depth_cart > max_depth)
    depth_cart = max_depth;
   else if (depth_cart < min_depth)
    depth_cart = min_depth;

   /* now convert these cartesian coords into array indicies */
   /* on 30 by 30 grid, (0 - 29) 15 - 15 is the center */
```

```
x_index = 15 - x_cart;
y_index = 15 + y_cart;

if ((x_index < 30) && (y_index < 30))
{ /* indicies are within array bounds */

 /* we now have the location of the contact in array indicies*/
 /* load the depth of the contact into the array using these indicies */

 if (left_depth_array_active)
 {  /* load into left depth array */
  left_depth_array[x_index][y_index] = depth_cart;
  if (depth_cart > *left_array_max_depth)
   *left_array_max_depth = depth_cart;
  else if (depth_cart < *left_array_min_depth)
   *left_array_min_depth= depth_cart;
 }
 else /* right array active */
 {
  right_depth_array[x_index][y_index] = depth_cart;
  if (depth_cart > *right_array_max_depth)
   *right_array_max_depth = depth_cart;
  else if (depth_cart < *right_array_min_depth)
   *right_array_min_depth = depth_cart;
 }

 } /* end if indicies in bounds */

 } /* if encountered contact */

} /* if not JUST_SCAN */

} /* end store_return_data */
```

```
/*+------------------------------+
 |
 |     up_arrays.c
 |
 +------------------------------+              */

/* Toggles depth arrays (active/not active) and reinitializes as necessary */

#include "sonar.h"
#include "gl.h"

update_depth_arrays(scan_complete, x_coord, y_coord, depth, left_depth_array_active,
 left_array_max_depth, left_array_min_depth, right_array_max_depth, right_array_min_depth,
 left_x_coord, left_y_coord, left_depth, right_x_coord, right_y_coord, right_depth)

 short *left_depth_array_active, scan_complete;
 int *left_array_max_depth, *left_array_min_depth, *right_array_max_depth,
  *right_array_min_depth;
 float x_coord, y_coord, depth, *left_x_coord, *left_y_coord, *left_depth, *right_x_coord,
  *right_y_coord, *right_depth;

{
 extern int left_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
 extern int right_depth_array[SONAR_RESOLUTION][SONAR_RESOLUTION];
 int i, j;

 if (scan_complete)
 { /* a full sweep has been completed */
  /* reinitialize inactive array */
  if (*left_depth_array_active)
  { /* switch to right_active and reinit right */
   *left_depth_array_active = FALSE;
   *right_array_min_depth = 99999;
   *right_array_max_depth = -99999;

   /* save AUV position with right array */
   *right_x_coord = x_coord;
   *right_y_coord = -1 * y_coord;
   *right_depth = depth;

   for (i=0; i<SONAR_RESOLUTION; ++i)
   {
    for (j=0; j<SONAR_RESOLUTION; ++j)
    {
     right_depth_array[i][j] = 99999;
    }
   }
  } /* if left active */
 else /* right array active */
 {  /* switch to left active, reinit left */
  *left_depth_array_active = TRUE;
  *left_array_min_depth = 99999;
  *left_array_max_depth = -99999;
```

```
 /* save AUV position with left array */
 *left_x_coord = x_coord;
 *left_y_coord = -1 * y_coord;
 *left_depth = depth;

 for (i=0; i<SONAR_RESOLUTION; ++i)
 {
  for (j=0; j<SONAR_RESOLUTION; ++j)
  {
   left_depth_array[i][j] = 99999;
  }
 }

 } /* if right active */

} /* if scan_complete */

} /* end update_depth_arrays */
```

```
/*+-------------------------------+
 |
 |     makefile
 |
 +-------------------------------+              */


/* This is the makefile */

CFLAGS = -Zg -lm
OBJS = disp_arrays.o
draw_numbers.o
edit_dials.o
edit_mask.o
edit_scline.o
edit_sweep.o
edit_tilt.o
edit_toggles.o
get_posit.o
get_sets.o
init_arrays.o
init_iris.o
load_data.o
main_sonar.o
make_arrows.o
make_chart.o
make_depth.o
make_dials.o
make_inst.o
make_mask.o
make_over.o
make_panel.o
make_plus.o
make_read.o
make_ring.o
make_rtn.o
make_scline.o
make_sweep.o
make_tilt.o
make_video.o
read_ctrls.o
read_sets.o
reset_flags.o
scan_ctrl.o
store_data.o
up_arrays.o

HDRS = disp_arrays.o
draw_numbers.o
edit_dials.o
edit_mask.o
edit_scline.o
edit_sweep.o
edit_tilt.o
```

```
edit_toggles.o
get_posit.o
get_sets.o
init_arrays.o
init_iris.o
load_data.o
main_sonar.o
make_arrows.o
make_chart.o
make_depth.o
make_dials.o
make_inst.o
make_mask.o
make_over.o
make_panel.o
make_plus.o
make_read.o
make_ring.o
make_rtn.o
make_scline.o
make_sweep.o
make_tilt.o
make_video.o
read_ctrls.o
read_sets.o
reset_flags.o
scan_ctrl.o
store_data.o
up_arrays.o

wesmar: $(OBJS)
    cc -o wesmar $(OBJS) $(CFLAGS)

$(HDRS): sonar.h
```

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chief of Naval Operations Director, Information Systems (OP-945) Navy Department Washington, D.C. 20350-2000	1
4.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	2
5.	Curricular Officer, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93943-5000	1
6.	Professor Robert B. McGhee, Code 52Mz Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	9
7.	Naval Ocean System Center Ocean Engineering Division (Code 94) ATTN: Paul Heckman San Diego, California 92152-5000	1
8.	Department Chairman, Code 69Hy Mechanical Engineering Department Naval Postgraduate School Monterey, California 93943-5000	1
9.	Professor D.L. Smith, Code 69Sm Mechanical Engineering Department Naval Postgraduate School Monterey, California 93943-5000	1

10.	Professor R. Christi, Code 62Cx Electrical and Computer Engineering Department Naval Postgraduate School Monterey, California 93943-5000	1
11.	Naval Coastal System Center Navigation, Guidance and Control Branch ATTN: G. Dobeck Panama City, Florida 32407-5000	1
12.	RADM G. Curtis, Code PMS-350 Naval Sea System Command Washington, D.C. 20362-5101	1
13.	Dr. and Mrs. A. A. Hartley R.R. 6, Box 592B Grove, Oklahoma 74344	1
14.	Mr. and Mrs. D. Avery 179 Rivo Alto Canal Long Beach, California 90803	1
15.	Mr. and Mrs. C. A Hartley 12703 Torrington Street Woodbridge, Virginia 22192	1
16.	Director of Research Administration Code 012 Naval Postgraduate School Monterey, California 93943	1
17.	Russ Werneth, Code u25 Naval Surface Warfare Center White Oak, Maryland 20910	1
18.	HQDA Artificial Intelligence Center ATTN: DACS-DMA, LTC Anthony Anconetoni The Pentagon Room 1D659 Washington, D.C. 20362-0200	1
19.	E. Scott Minner Undersea Systems Department General Electric Company P.O. Box 4840 Syracuse, New York 13221-4840	1